Design and Implementation of a Portable Neural Network Interpreter

Joan Campbell-Tofte

A thesis submitted to the IT University of Copenhagen
for the Degree of Master of Science in IT
(the Software Development Area)

Supervisor: Professor Peter Sestoft

Commissioned by: Anders Gorm Pedersen
CBS, DTU.

November 9, 2002

Abstract

This report describes the design and implementation of a Neural Network Interpreter to be used for the simulation of artificial neural network-backed predictors designed to analyse biological data. The application was constructed using the component-based software engineering process in the object oriented paradigm and is written in Java. The resulting software is a portable standalone application, that works through a graphical user interface and can simulate several neural network-predictors that address very different problems. The Neural Network Interpreter is designed specifically for the neural network-backed predictors developed at the Center for Biological Sequence Analysis of the Technical University of Denmark. The graphical user interface for the interpreter is constructed dynamically by reading input files which specify the predictors, the so-called *parameter files*. The parameter file contains among other predictor variables, the list of synapse files that specify the artificial neural networks used in the predictor. Extensibility is thus built into the software because all that is required to add newer predictors is for the user to save the parameter files for the new predictor in the Interpreter's input source directory.

Contents

Preface with Acknowledgments

The work described in this Master of Science thesis was carried out at the IT Univerisity of Copenhagen (IT-C) in 2002, under the supervision of Professor Peter Sestoft, Department of Mathematics and Physics at the Royal Veterinary and Agricultural University in Denmark and at IT-C. The idea for construction of a Neural Network Interpreter (NNI) came from a suggestion by Anders Gorm Pedersen, Associate Professor, Center for Biological Sequence Analysis (CBS), of the Technical University of Denmark. The thesis submission includes a CD-ROM containing the source code for the NNI, example parameter files and their associated synapse files, a user manual and example input test data. These are also available for download at ⟨http://www.it-c.dk/people/joan/⟩.

Firstly, I will like to thank my supervisor, Professor Peter Sestoft for his excellent supervision and guidance so that I finally grasped the logic of programming. I would also like to thank Anders Gorm Pedersen for suggesting a very interesting project and providing all the background material required for building the predictors. Special thanks go to Chigbo Uzokwelu, Sussana Pedersen, Nada Alzubeidi, Kevin Hechtl, Hai Son Nguyen and Asem Butt, my friends at ITU, for patiently listening to my long tales about trying to implement the interpreter and for helping me with all sorts of problems. Special thanks also go to Birgitte Tofte for painstakingly reading and correcting language errors in the report. Heartfelt thanks go to Bose and Osi, my little sister and brother and Franca Ikhuenobe for being such angels in a very difficult time. Finally, many thanks go to my husband Mads, for his inspiration and unflinching support all through the process. Special thanks too Mads, for guiding me through the maze of LaTeX and Concurrent Version System; I know it was not easy to 'keep the house tidy.'

Joan Campbell-Tofte November 9, 2002
The IT University of Copenhagen
joan@itu.dk

Chapter 1

Introduction

This project involved the design and implementation of a Neural Network Interpreter which should simulate trained artificial neural network-based predictors used for analysing biological data. In this chapter, the background to the problem domain and the problem formulation are presented. In addition, the motivation for the project, as well as its scope and limitations are stated.

1.1 Project Background

The blueprint for all living organisms resides in their deoxyribonucleic acid (DNA) content. Therefore, in carrying out comparative studies of DNA and the proteins encoded within, it is possible to make far reaching conclusions about the origins and relationships among living things. Within the last 10 years, technological advances in the determination of genomic sequence data (from DNA) and proteonomics have resulted in a huge output of biological data. The data is accumulated so rapidly, that it is impossible to keep pace with performing the experimental analysis that is required to make meaning of all the data. This development has increased the need for computational analysis, especially involving the use of predictors for mining the information available in the accumulated data. A good example for a biology problem where computer analysis is invaluable is in the determination of protein sequence from studying the genetic material (DNA) that codes for it. Although it is true in theory that the DNA coding for a given protein can be inferred from the protein's sequence, in practice with DNA from animals and plants, the DNA that codes for protein is never in a continuous stretch. There is always some extraneous DNA (the so-called introns) interspaced between the actual coding regions. Since it is difficult to sequence the proteins directly, as opposed to DNA sequence determination which has become automated, for a lot of proteins, biologists have to rely on obtaining the DNA sequence and then using computer programs to predict which parts of the DNA sequence codes for proteins. Indeed, a whole new field, Bioinformatics has emerged from all activity involving biocomputing.

Aside from the obvious advantage that we will learn a lot about the origins and evolution of the species, bioinformatics holds great promise for future significant advances in several aspects of biomedicine. For example, with the recent full mapping of the human genome [2], physicians will be able to use genetic information to diagnose diseases as most medical conditions have one or more genetic underlying components. Apart from identical twins, no two individuals have identical genetics. Hence, the human genome information provides a good basis for drug targeting, since more specific drugs can now be developed, and treatments can be individualized. In health care and family planning, a better understanding of our genetics will help to avoid the passage of certain inherited diseases to the next generations. In forensic medicine, newer and more specific methods are being developed for identifying suspects or victims using DNA pattern information.

At the Technical University of Denmark, Lyngby, the Center for Biological Sequence Analysis (CBS) has pioneered the development of several software tools for the analysis of both DNA and protein sequences. Applications of the computational tools or predictors include, classifying the biological sequences, separating protein coding regions from noncoding regions in DNA sequences, predicting molecular structure and function, reconstructing the underlying evolutionary history, etc. All of which have become an essential component of the research process within academic and industrial biotechnology. Although the service is freely available from their CBS website at ⟨http://www.cbs.dtu.dk/services/⟩, it routinely happens that a user submits a sequence and has to wait for long periods of time to get the results of the analysis. This is due to the limitations on the number of sequences that can be received and processed by the online servers at any time. For the researchers, valuable time could be saved if they had the predictors installed as a local standalone application. Aside from the time considerations, there are some biotechnological companies that are nervous about submitting 'precious' sequences on the Internet for fear that unscrupulous people could snap them up. Thus, the use of the prediction servers via the Internet poses a potential for patenting problems for people with commercial interests. Again, such companies would be better served with in-house versions of the predictors. CBS is currently distributing the predictors in the form of a collection of scripts and executables that are accessed through a web browser, essentially using HTML to build a graphical user interface. However, this arrangement requires that the user has a web-server set up. Furthermore, as most of the CBS programs are written in Fortran, there is often the need to customise some of the accompanying scripts to the local platform (mostly UNIX-like operating systems). The idea behind this project is to solve the problem by replacing the suite of Fortran programs with a single program, the so-called Neural Network Interpreter (NNI), which we have designed.

1.2 Problem Formulation

An interpreter can be defined as a program that takes another program and its input data as input (to the interpreter), executes the input program's instructions on input data and produces as output (from the interpreter), the output from the simulated program. Eighteen out of the twenty CBS prediction servers are based on artificial neural networks (ANNs). ANNs are collections of mathematical models that are loosely based on the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. Learning here implies that each of the CBS neural network-prediction servers has been written and trained to tackle a particular problem. For example, NetStart World Wide Web server (refer the CBS page) can only effectively predict translation start sites (i.e. starting point for protein synthesis) in plant DNA, while Promoter2.0 can only predict transcription start sites (starting point for RNA synthesis) in vertebrate DNA[1]. The question then for this project is:

> **Can a platform independent Neural Network Interpreter that effectively generalizes and simulates each of the different CBS artificial neural networks be constructed?**

In constructing interpreters, one has to distinguish between the meta- and object-languages. The meta-language is the language in which we write the interpreter, while the object-language is the language of the program that is evaluated by the interpreter. In the existing neural network-based predictors, the information for the neural network's architecture, weights, window size and alphabet lies in text files known as synapse files (see Section 2.2.8 for an example). This means that there is a specific synapse file for every ANN. In order to achieve the level of generalization required for the Neural Network Interpreter that is envisaged in this project, our solution is based on a concept we have named the *parameter file*. The parameter file is

[1]Readers that are unfamiliar with biological terminologies may refer to Brazma et. al. [1] for a brief introduction to the field of molecular biology.

also expressed in simple text file format and it specifies all the variable properties of the ANN-based CBS predictors. More specifically, the parameter file defines:

- The names of the synapse files for the network(s) used in the particular predictor.
- The nature and manner of data analysis within the ANN. For example, the contents of the parameter file indicate whether the network will only analyse data from around a specific substring, or if all of the data will be analysed. Similarly, the parameter files also specifies if the networks accept only full or also accepts partially full windows of input data. (see Section 4.3.1 for an example of a parameter file).
- The way of deriving the final predictor output from the results calculated in the different networks. With the CBS ANN predictors, the output is calculated either by taking an average or making a juried decision of the output from the different networks within the predictor.

Therefore, in our solution, the input to the interpreter (i.e. both the predictor, made up of parameter file and its associated synapse file(s), and test data), are all in simple file format. For the meta-language, we have chosen to implement the Neural Network Interpreter in the Java programming language. Thus, while the introduction of the parameter file simplifies the object-language for the interpreter, the use of Java as meta-language ensures that the interpreter can be used on any hardware with a Java virtual machine (JVM). The JVM is available freely from the website ⟨ http://java.sun.com/j2se/1.4.1/⟩. An added advantage to the proposed solution, is that there is now only one piece of software to be maintained.

1.3 Motivation

Coming from a background in Biology, it was of great interest to me to carry out an M.Sc. project which would involve applying the principles and tools for software engineering that I have learned at the IT University, to solving a contemporary problem for biologists. As the proposed application should be portable between different platforms, I have chosen to implement it in Java, an object-oriented language that was designed for development of large systems from reusable components. In addition to implementing the software in Java, the project has also allowed me to practice the technical concepts, methods and measurements of object-oriented programming during the development process, something that was important to me as I wanted to become a proficient software developer.

This thesis has thus involved the design and development of a Java language-based application which is platform independent and should interpret any of the neural network-based sequence analysis prediction servers available at the CBS home page.

1.4 Tools for Software Development

There are several Computer-Aided Systems Engineering (CASE) tools that should help in the different phases of the construction of reliable OO software (refer to the web site ⟨http://www.cs.queensu.ca/Software-Engineering/case.html⟩, for a summary of examples of such tools). For a smaller application like the NNI and in order to properly acquaint myself with the entire software development process, I considered that it was more profitable for me to generate the different work products by hand. There were therefore, no special support tools used in the analysis, design, implementation and testing of the NNI. The techniques employed include:

- Extraction of system specifications for the proposed software.

- Translation of the information into use cases and representative classes, using the Class-Responsibility-Collaborating modelling.
- Static and dynamic modelling of the resultant classes using the UML class, event and activity flow diagrams.
- The program was written in Java on Windows and this report was written in LaTeX on Linux.

A standard Java editor (JCreator 2.5) available from ⟨http://www.jcreator.com⟩ was used as editor. The charts and illustrations were drawn using dia, a drawing program available from ⟨http://www.lysator.liu.se/ alla/dia/ ⟩. Dia files were saved as Encapsulated PostScript (EPS), making it possible for them to be scaled arbitrarily to any size without loss of resolution and for incorporation in this report. Version control was achieved by storing source code in folders labelled with the date of creation. Late in the development process, my home-made version control caused me so much confusion that I had to switch to using the Concurrent version control system.

1.5 Scope and Limitation

The CBS neural network-based prediction servers are all of a particular architecture (backpropagation trained multilayer perceptrons; for full descriptions, see Chapter 2. They have all been trained to tackle problems in protein or nucleic acid data. Therefore, the Neural Network Interpreter described in this work should only interpret CBS-type network-backed predictor(s) and only accept protein or DNA sequences as input test data.

1.6 Reader's Guidance to the Document

This report is written first and foremost as a dissertation for a Master of Science degree in Information Technology (Software Development Area), at the IT University of Copenhagen. Therefore, the report is directed at my supervisors and external examiner(s). I also envisage that those who wish to use, maintain and modify the Neural Network Interpreter described herein, will find this report useful.

In Chapter 2, relevant terminology concerning bioinformatics and machine learning methods is introduced. In addition, the composition, architecture, types and training of artificial neural networks are presented. In Chapters 3-6, the considerations and steps taken during the developmental phases for the software are outlined and discussed. The phases are specification and analysis of the application requirements, design, implementation and testing of the software, respectively. Although I started off with requirements specifications and went on to design the system, during implementation and testing, both the system specifications and design were modified iteratively. As such, the Neural Network Interpreter's requirements specifications, design use cases, class and design diagrams presented in this report, represent the final versions of these documents. The Chapters 3 to 6 should appeal to those who wish to maintain and modify the application. Chapter 7 is a user manual directed at a typical user of the system who is not necessarily interested in understanding how the system is put together. In Chapter 8, the report is concluded with an evaluation of the software development process and the work products generated in the course of the project.

Chapter 2

Artificial Neural Networks in Bioinformatics

In exploring how neural network-based predictors work, this chapter starts off with an introduction to why machine learning approaches are used within the field of bioinformatics. Next, the architecture, training, evaluation and data representation for the artificial neural networks (ANN) are described. Finally, the advantages and disadvantages with using ANNs are reviewed.

2.1 Bioinformatics and Machine Learning Approaches

As mentioned in Chapter One, Bioinformatics is a new interdisciplinary research area which comprises the application of computers for the analysis of biological sequence data. There are 2 major aspects to bioinformatics. First, a vast amount of sequence data is generated and deposited in central databases by thousands of research teams working in biological and chemical laboratories all over the world. The best known biological sequence databases are SWISS-PROT (for proteins, accessible via ⟨http://www.ebi.ac.uk/swissprot/access.html⟩) and GenBank (for nucleic acid sequences, accessible via ⟨http://www.ncbi.nlm.nih.gov⟩). Next, the data are analysed by computer driven methods for extrapolation of biologically important information. Such analyses include:

- Mining of biological sequence for information (e.g. prediction of translation start sites).
- Recovery of evolutionary patterns.
- Prediction of gene content and function in newly sequenced genomes (a good example being the recent determination of the entire human genome[2]) and lately,
- Silicon-based biology for simulation of entire biological systems rather than focusing on individual constituents.

A basic feature of the biological macromolecules (Deoxyribonucleic acids or DNA for short, and proteins) is that they can be represented as strings built from a set of symbols taken from a short alphabet. While DNA consists of strings over the alphabet ACGTX, protein sequences are made up of amino acid residues denoted by the alphabet ACDEFGHIKLMNPQRSTVWYX. The X in both alphabets is used to represent the unknown residue, i.e. it could be any of the 4 (for DNA) or the 20 (for proteins). Thus, it is easy to cast them in the form of character strings for computer analysis. Often the object of many computer driven methods is to assign function to a biological sequence data or to classify a given sequence in one of a number of categories. In the algorithms traditionally used to analyse the biomolecules, related sequences from different species are first aligned and compared for conserved sequences using the global [3] and local [4] pairwise alignment algorithms. These algorithms and their improvements with the introduction of the theory of local alignment scores and substitution matrices, have made it possible to effectively assess the statistical significance of the sequence similarities across species [5].

However, in matching sequences across species, it may be difficult to find any resemblances within closely related species, that are evolving at very different rates. On the other hand, spurious close relationships may be observed among converging widely differing species. Therefore, the resulting sequence models must at best, be probabilistic [6]. For the biological data, a different group of computer methods for recognizing and extracting patterns exists. Unlike the methods that are based on sequence alignments, the sequences themselves are used as input data. Here, the prediction methods improve through 'experience' or 'learning' on exposure to the input data. Any relationships between the sequences are therefore calculated directly from the examples without *a priori* knowledge about the biological significance of the sequence patterns. Hence the methods are described as data-driven or machine learning approaches [6, 8].

These machine learning methods are very well suited for areas where there is a large amount of data but little theory. In answer to the question as to why machines should be designed to learn as opposed to being designed to perform precisely as desired in the first place, Nilsson [11] has listed the following engineering advantages:

- Some tasks are best defined by examples. That is, we can specify correct input and output data, but not the type of relationship between them. For example, the stretch of DNA at the beginning of expressed genes can be experimentally determined for several genes in a species. However, it is not often possible to deduce all the salient cell signals within the sequences by simply aligning the nucleotides. This is because different spatial arrangement and content of the significant residues can produce the same 3-dimensional effect in different genes from the same species.
- When dealing with large quantities of data (e.g. whole genomes), there may be hidden correlations and relationships that are much easier for the machine to extract, as compared to a human.
- Certain characteristics of the working environment might not be known at the time of the design. Hence machine learning can be used for on-the-job improvements.
- Environments change with time, therefore machines which adapt to changes would naturally reduce the need for redesign.

Machine learning approaches have a wide range of applications in areas other than bioinformatics. Such applications include automatic recognition of patterns with human faces or speech, prediction in commerce (e.g. predicting the stock market or customer behaviour), in operations of engineering systems, datamining, or in internet-related problems (e.g. in spam filtering, searching, sorting, etc.). Apart from the artificial neural networks, machine learning approaches also include statistical methods (e.g. Hidden Markov models) and adaptive control theory. As my project is mainly concerned with ANNs, I shall limit my considerations of machine learning algorithms to the artificial neural networks for the rest of this report.

2.2 Artificial Neural Networks

2.2.1 Composition of ANNs

Artificial neural networks (ANNs) are defined as non-linear computational elements, interconnected through adjustable weights. Each unit is referred to as a 'neuron' (refer Figure 2.1). The output from a neuron which

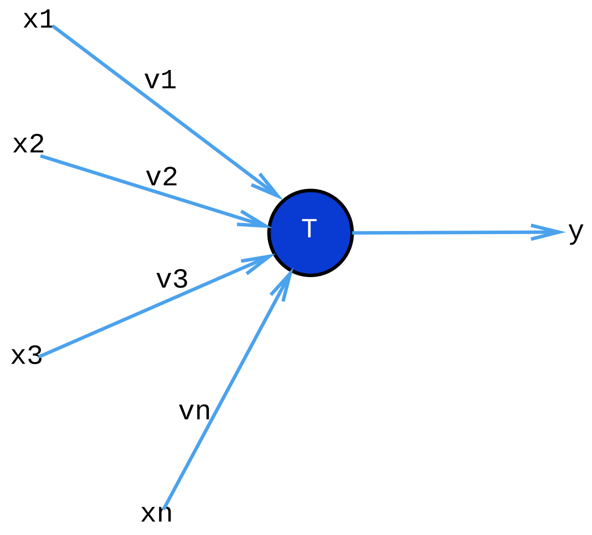

Figure 2.1: A formal neuron showing input $x_1, x_2, \cdots, x_n$, weights $v_1, v_2, \cdots, v_n$, and threshold value T. The output y is 1 if $x_1 v_1 + x_2 v_2 + \cdots + x_n v_n$ is greater than T, and 0 otherwise.

is either on or off, depends solely on the inputs. The neuron receives a number of inputs, performs a very simple computation which involves multiplying each input by an associated value known as the weight, and sends out an output if the weighted sum of inputs exceeds another predetermined value known as the threshold. Thus, an efficient unit which sends out signals must have correspondingly larger weights as compared to the passive neuron. The threshold value is either implemented in the form of a so-called 'bias' value, which is added to the weighted input sum or as the threshold value that is subtracted. When the sigmoid function illustrated in Figure 2.2.1 is applied to the weighted sum of inputs plus the bias value, the activation of the neuron becomes non-linear, scaled and continuous between 0 (for $x \to -\infty$) and 1 (for $x \to \infty$). Generally, the threshold and weight values are unique for each neuron. Indeed the neuron's weights which are derived during the learning or network training process constitute the neuron's special contribution to the ANN. In summary, the artificial neural network consists of a collection of neurons. The way in which the individual neurons are put together determines the architecture of the network. Note that there is no general rule for what network architecture is best for solving any given problem [7].

2.2.2 Learning in ANN Versus *a priori* Problem Solving

As with conventional algorithms, an input is presented via the input neurons to the network, this input is propagated through the whole network and finally, some output is produced. Hence, ANN can be thought of as functions. The input and output can consist of many components, and as such, networks also map vectors to vectors. However, ANNs differ from conventional computer programs in that the analysis is not based on a set of a logical sequence of rules, instead the neural networks are trained by a presentation of examples. As such, the details of how to perform the task are not needed. What is needed is a set of examples that is representative of all the variations of the task. The examples however need to be selected very carefully if the network is to perform reliably and efficiently.

Computation in ANNs is solely due to a dynamic process of firings from the neurons. The high connectivity of the network (i.e. the many terms which contribute to the sum of the weighted inputs), has two major effects. Firstly all functions are infinitely continuous so that, small changes in the results can be achieved by

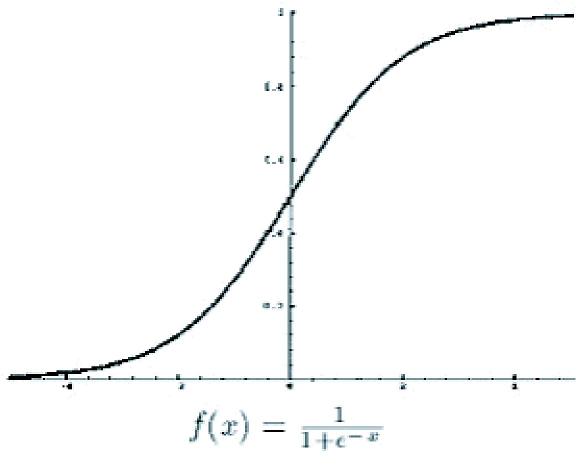

Figure 2.2: The sigmoid activation function

very small changes in the input. Secondly, inconsistencies or disturbances will have local effects, suggesting that the system can be robust and that its performance will be maintained even in the presence of noise or errors in input data [9]. This robustness in the face of noise and errors is what makes networks well suited to pattern matching. Although ANN cannot do anything that cannot be achieved using conventional computing techniques, they are better at complex functions involving pattern recognition once trained for that purpose.

2.2.3 Short History of the Development of ANN

Historically, the concept for the artificial neural network was first introduced in 1873 by Alexander Bain in his book entitled ”Mind and Body. The theories of their relations”; (cited in [12]). The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits (cited in [9]). Although the concept of ANN was inspired by biological neurons, the artificial neurons are so simplified and generalised that any resemblance to neurobiology is merely of historical importance. Nevertheless, there are still some parallels in the terminology used both with ANN and in biological neurons. For example, the strength in both systems is considered to come from the parallel distributed processing which the simple neurons taken together provide. In addition, the weights in ANN are thought to be analogous to the synaptic connections between biological neurons. Finally, the ’learning’ process achieved in ANN via adjustments to the neuronal weights is also considered to be similar to learning in the biological system.

ANNs went into disfavor when Minsky and Papert published their book (Perceptrons, MIT Press, 1969; cited by [14]), which proved that the perceptron with a unit step activation function could not solve simple linearly inseparable problems such as XOR. However, since the 1980s, ANNs have enjoyed a revival with the development of more sophisticated algorithms to support the networks as we know them today. These involved the introduction of networks built from interacting multiple layers of neurons, thresholding with non-linear functions and a new learning rule known as the ‘backpropagation rule’ [14].

2.2.4 Types of Artificial Neural Networks

ANNs are classified into different types based on their architecture, the way data is fed through the network and on their mode of learning or training. The more powerful networks types (in terms of their analytical capabilities) and their properties are as follows [10]:

- Multilayered feedforward Perceptron.

  In the multi-layer perceptron (MLP), the neurons are organized into input, hidden and output layers as shown in Figure 2.4 (refer Section 2.2.8) below for a sketch of a two-layered ANN. There are no conventions for counting the number of layers in the network but Hertz et al. [23] recommend that an n-layer network has n layers of connection or weights and n-1 hidden layers.

  Information flows from the input layer to the output layer, via the hidden layer, without any feedback connections. Hence they are said to be of the feed-forward type. The units in the middle layer are described as hidden because they have no direct connection to the outside world. There can be several hidden layers. CBS networks usually have only one hidden layer as this is sufficient to represent any boolean function (Personal communication, Anders Gorm Pedersen). The term input layer neurons is a misnomer, since the sigmoid function is not applied to the output value from these neurons. Their raw values are fed directly into the layer downstream the input layer (i.e. the hidden layer). The output (H_j) from the j^{th} neuron in the hidden layer is computed using the formula:

  $$H_j = \sigma(\sum_i v_{ji} I_i + t_j),$$

  where v_{ji} is the weight of the connection from the i^{th} input neuron to the j^{th} hidden neuron, I_i is the corresponding signal from the input layer, t_j is the bias value assigned to the neurons in the hidden layer, and σ is the sigmoid activation function.

  Similarly, the output (O_k) from the k^{th} neuron in the output layer is computed using the formula:

  $$O_k = \sigma(\sum_j v_{kj} H_j + t_k),$$

  where v_{kj} is the weight of the connection from the j^{th} hidden neuron to the k^{th} output neuron and t_k is the bias value for the neurons in the output layer. In this way, the data fed into the network as encoded sequence data is propagated forward to generate the output which may typically represent some structural or functional features [6].

  The manner in which the neurons in an upper layer are connected to the neurons in a lower layer is described as the connectivity of the network. A standard form of connectivity between network layers (illustrated in Figure 2.4), is one in which every unit in one layer is connected to the units in the following layer. MLPs are usually trained using the supervised backpropagation method (described in detail below in Section 2.2.5). As all of the CBS networks are of the MLP type, the Interpreter described in this thesis is dedicated solely to simulating MLPs.

- Hopfield or feedback networks.

  The Hopfield type of neural network was introduced by J. J. Hopfield in 1982. It consists of a set of neurons that are connected to each other. Hence there is the possibility of sending information from any one unit to other neurons all over the network and there is no grouping of the units into input and output layers. Furthermore, the network training is unsupervised, meaning that the training data is not pre-classified by the trainer. The network classifies the data as they are presented.

- Kohonen Feature map.

  The Kohonen Feature map is also composed of unlayered neuronal units that are arranged in a rectangular grid. In contrast to MLPs and like Hopfield networks, the Kohonen Feature maps take only input values, which are then classified by the network itself during training.

2.2.5 Training Networks

In this section, only the supervised training of MLPs will be discussed. As indicated earlier, training ANNs to tackle a particular problem or its learning process, refers to the process of adjustments in the weights connecting the neurons. The direction and magnitude of the changes are determined by how the networks perform when exposed to the known values. Adjusting weights during training is hindered in MLPs because it is difficult to estimate the errors in the hidden layer. This is where the advantage to the developments in ANN by the introduction of the training algorithms comes in. In MLPs, the most commonly used training algorithm is called the backpropagation of errors algorithm because the error observed in the output layer is backpropagated to the previous layer. The steps involved in the backpropagation of errors in a network as outlined by [23] and [8] can be summarized as follows:

1. First, a training set consisting of a set of data points (sequences), each of which is labelled with the known output category is fed into a naive network, and the output generated in response to the input data is noted. In CBS, the naive network weights are set to random values typically between $+1$ and -1.

2. The weights are altered in a series of steps, to reinforce correct decisions and discourage incorrect ones. That is, the error in the observed output as compared to that expected is reduced systematically as follows:

   - First, the squared error in the network output or the *quadratic error measure* E is calculated from the equation:
     $$E = \sum(O^k - T^k)^2$$
     where the Ts represent the expected output values from the training input data and the Os represent the observed output values obtained in the output neurons. Note that because the error function implicitly assumes a normal distribution of target values, other error functions are sometimes found to be more useful especially with classification [6].
   - Then starting with the output layer, the weights are adjusted using the gradient descent algorithm. For this purpose, the error terms for each of the output neurons is calculated and used to compute the weight adjustments Δv_{kj} represented in the differential equation:
     $$\Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}}$$
     where η is termed the learning rate. The latter is a measure of how much the weights are changed in one step of the network training phase. A low learning rate or small η is more desirable as this ensures a more stable convergence of observed output and expected output values.
     If the error measure and activation function of the neurons are differentiable, then
     $$\frac{\partial E}{\partial v_{kj}} = 2C_kH_j$$
     where
     $$C_k \equiv O_k(1-O_k)(O_k - T_k).$$
     This implies that
     $$\Delta v_{kj} = -\eta 2C_kH_j.$$

Therefore, the adjustment in the weights of the output layer can be expressed as a function of expected output neuron value, O_k, observed output values T_k, inputs from the hidden layer H_j and the learning rate η.

In like manner, the adjustment in the weights of the neurons in the hidden layer can be expressed as a function of input and output values in this layer, and as a function of the error in the output layer using the equation:

$$\Delta v_{ji} = \frac{-\eta \delta E}{\delta v_{ji}} = -\eta 2 D_j I_i;$$

where

$$D_j \equiv H_j(1 - H_j) \sum_i C_k v_{kj}.$$

Thus, the training algorithm propagates the error committed in the lower output layer backwards within the network, hence the name backpropagation.

3. The whole sequence of exposure to test data, calculation of error terms and adjustments of weights, continues until some stop criteria representing convergence between target vectors and output values has been met (See Section 2.2.6 below for these convergence criteria).

The software constructed in this project will not carry out training of ANNs. It will therefore only handle synapse files of trained networks.

2.2.6 Evaluating ANN Performance

The endpoint for training an ANN is difficult to determine, because one risks ‘overtraining’ the network as the error measure approaches a minimum. Overtraining describes the situation where the weights are such that the network has acquired an almost perfect fit for the training examples. The ‘memorising’ of the training suite is acquired at the expense of the desired generalization ability required when the network is presented with new input data. Overtraining can be counteracted by testing the network during training, with a set of data that it has not been exposed to. A standard practice is to set aside a set of unknown data analogous to that used in training. This data set is known as the test set. At intervals in the training process, the prediction performance of the network is measured against the test set. The ANN is considered trained when its performance with the test set peaks.

There are several measures for ANN performance. For networks that are trained to tackle classification problems, a *winner-takes-all* approach is applied by thresholding the output neurons. This converts the output to a ‘yes/no format’. The prediction performance of the ANN on two-category problems can thus be determined using the Matthews coefficients [23, 24]:

$$C = \frac{(P_T * N_T) - (N_F * P_F)}{\sqrt{(N_T + N_F)(N_T + P_F)(P_T + N_F)(P_T + P_F)}}$$

where:

P_T is the number of true positives.
N_T is the number of true negatives.
P_F is the number of false positives
N_F is the number of false negatives.

The Matthews coefficient expresses the degree of correlation between the two categories and assumes unit value if all the test examples are correctly classified and 0 if there is no correlation between input and output values. Negative correlation values can be obtained if there is an inverse relationship between the prediction and the true value. Hence the measure is symmetric and does not favor positive or negative examples. This makes it preferable to the performance measure based on percentage of correctly classified examples.

ANNs that have too many parameters (neurons, weights and bias values) are also at risk of overtraining. The parameters are described as being too many when they outnumber the number of examples used to train the network. A rule of thumb is to set the number of parameters anywhere from between half as many training examples, to not beyond the total number of examples [23].

2.2.7 Encoding Input Data for Artificial Neural Networks

Another important aspect of using ANNs is finding good representation for the input data. The data to be encoded can be extracted from single positions, regions or whole sequences. The standard encoding technique used for the CBS neural networks is the *sparse encoding*, where each symbol in an alphabet of n letters is represented by n input values. The specificity comes from representing one of the values as being ‘on’ (or 1), while the rest are ‘off’ (or 0). For example, sparse encoding the characters in an alphabet made up of the set ACGT would mean that :

$$A = 1000, C = 0100, G = 0010, T = 0001.$$

For the set ACGTX, sparse encoding would result in:

$$A = 10000, C = 01000, G = 00100, T = 00010, X = 00001.$$

Therefore, sparse encoding ensures that the number or active neurons (or units capable of firing) in the stored patterns is small, as compared to the number of units in the network. Once encoded, the data is presented to the network as a moving window containing the position to be evaluated at the center of the window, flanked by the surrounding sequences.

Indeed, the encoding method is not trivial and has been found to greatly influence the quality of the network’s results [15]. Medler and McClelland [16] found that the combination of context and sparse coding constraints increased network accuracy and encouraged the formation of more robust internal representations, especially for larger data sets. This is because the approach lets the network do what it does best, i.e. extract the relevant information from the raw data itself. There is however the drawback that the number of parameters required in the network increases dramatically with larger alphabets. This is especially critical when working with protein sequences where the alphabet consists of twenty or twenty-one characters. As indicated in Section 2.2.6, a careful design of network parameters in the ANNs can prevent such problems. For example, the window size (defined as the number of residues examined at one time by the network) can be made smaller in a network using a large input alphabet.

2.2.8 The CBS Neural Network-based Predictors

In this section, the set–up of typical CBS predictors is used to illustrate the properties of the ANNs that have been earlier discussed. NetStart 1.0 Prediction server is an example of a CBS web-based neural network-backed predictor, that has been trained to indicate translation start sites in vertebrate and *Arabidopsis thaliana* (plant) cDNA sequences [13]. Like the other CBS predictors, it is freely available from the URL address, ⟨http://www.cbs.dtu.dk/services⟩. The NetStart page (refer Figure 2.3) presents the user with graphical user interface carrying notes on what the predictor can do, as well as precautions one must take for obtaining optimal results. For example, the NetStart user is informed that the network was trained on predominantly cDNA sequences, that is DNA sequences that have been derived from messenger RNA.

Figure 2.3: Graphical user interface of NetStart 1.0 predictor

This means that cDNA contains only gene coding sequence, without the extra DNA (or introns) which interspace coding sequences in eukaryotic DNA. The warning is therefore of utmost importance as applying the predictor to nucleotides containing introns would give totally wrong results.

Input test data are submitted either as pasted sequences in a form provided in the NetStart graphical user interface (GUI) or by using the browser to access the necessary file. In both cases, the file has to be submitted in the FASTA format. A sequence in FASTA format [27] begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column. For a more detailed description of FASTA format, see ⟨http://www.ncbi.nlm.nih.gov/BLAST/fasta.html⟩. The use of the format ensures that there is a standard in input sequences, as well as providing a label for the input data that can be used as reference to the particular input data within the software. The page also provides links to a user manual, an explanation of the result output and an abstract to the article describing the predictor. Note that NetStart is totally independent of the other predictors. Thus, the code used for generating the user interface, the underlying network(s), as well as that code for controlling the analysis, are specific for the NetStart predictor.

As mentioned in Section 1.2, the synapse file contains the blueprint for the network. It is therefore relevant at this point to outline a typical CBS synapse file. I have chosen as the example, the synapse file (MySyn.txt) for a small network specially built by Anders Gorm Pedersen for testing the Neural Network Interpreter that was developed in this project. MySyn.txt's weight parameters were kept to a minimum number to facilitate my carrying out all the calculations for a short input sequence by hand. Hence, it is very simple and its list of weights is short enough for closer examination. It differs from the typical CBS predictor synapse file (NetStart, for example, which can have a thousand or more weight parameters) only in the length of the list of weights. The MySyn.txt network was trained to predict the presence of G (guanine) residues in DNA and it describes the architecture of the multiple layer perceptron illustrated in Figure 2.4.

Synapse File for Test Network: MySyn.txt

```
TESTRUNID          IN: ACGT  OUT: i.
      12  LAYER:   1
       2  LAYER:   2
       2  LAYER:   3
     100 :ILEARN        3 :NWSIZE      -90 :ICOVER
     -0.22020     -0.24010     -0.17422     -0.25131     -1.36468
     -1.55053      3.48135     -1.45199     -0.35687     -0.48446
     -0.26066     -0.18385     -0.68584     -0.11751      0.03703
     -0.00989     -0.05937     -1.17373     -1.15452      2.46427
     -1.08576     -0.01796     -0.12046     -0.15830     -0.25301
     -0.74973      3.26438      2.19347     -2.59703     -3.46976
     -1.92930      2.57139
```

The synapse file starts with a string identifying the latest training run (typically, the default value "TESTRUNID"). This is followed by the input alphabet (in this case 'ACGT') and the output alphabet ('i' and '.'). The short 4 symbol alphabet indicates that the network is trained to handle only known DNA sequences (i.e. only nucleotide sequences without unknown residues will be correctly handled by the network). Furthermore, the total size of the alphabet indicates that each letter in the input sequence will be coded into a string of four binary digits, only one of which is set (in the so-called 'sparse encoding', refer Section 2.2.7 for rationale behind data encoding in ANNs). When the network has analysed test data, the result can be presented in text format, with the letter 'i' representing the point on the input sequence where

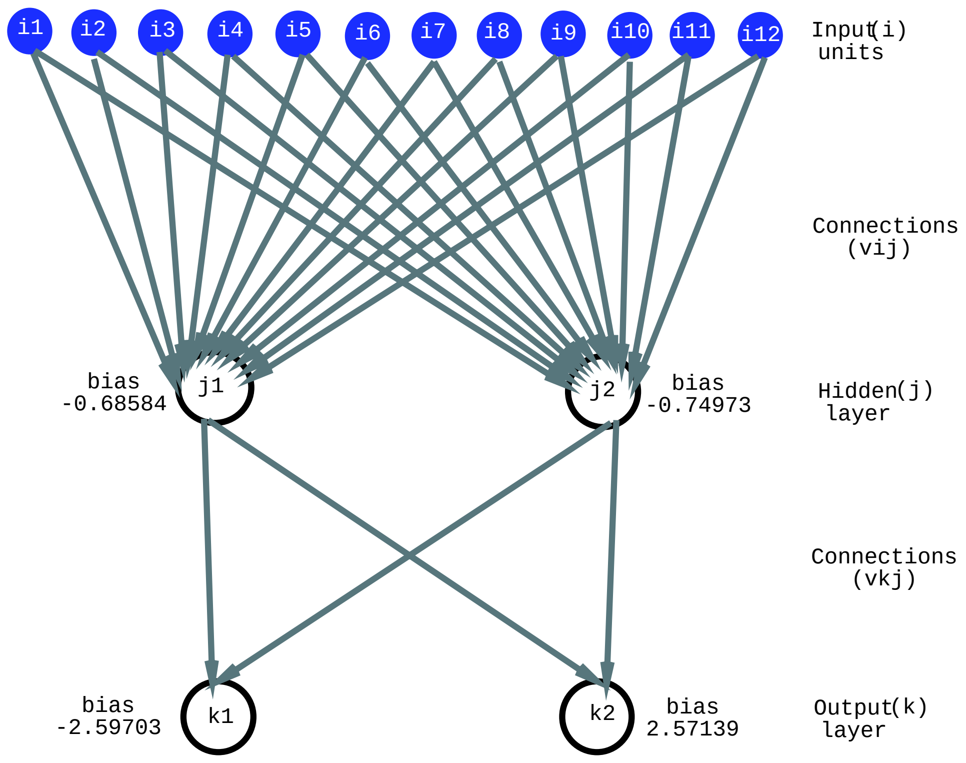

Figure 2.4: A sketch of two-layered artificial neural network specified by the synapse file, MySyn.txt, showing the input, hidden, and output units or neurons (i, j, k, respectively) and their connections (v_{ij} *and* v_{kj}). The values which are indicated beside the hidden and output neurons correspond to their bias values.

there was a more than 50% chance that there is a G residue, while the '.' signifies that the prediction was lower than 50%. For example,

> on analysing an input sequence 'ACAGCTA', the network should produce the result '...i...'.

The next three lines specify the number of processing units (neurons) in the input (marked 1), hidden (marked 2) and output (marked 3) layers, respectively. From the numbers preceding the word 'LAYER', we can see that there are 12 neurons in the input layer, 2 hidden neurons, and 2 output neurons. The number placed before the term 'ILEARN' refers to the number of training cycles used to construct the network. One training cycle implies that all examples in the data set have been presented to the network. Thus it took 100 training cycles to train the network. The window size for presentation of data to the network is specified by the word, 'NWSIZE'. Therefore, in the predictor represented by the synapse file- MySyn.txt, the input data consists of three nucleotide residues, and explains the need for 12 input units (i.e., 3[window size] times 4[total number of input alphabet]). The 'ICOVER' value indicates the 'size of receptive field', a now redundant feature that was used to constrain the values of the weights in the input layer (personal communication from Anders Gorm Pedersen).

The rest of the synapse file is taken up by the weights and bias values attached to each of the network neurons. The order of numbering starts off from the leftmost hidden units. In accordance with an input window of 12 units, the first 12 numbers in the synapse file correspond to the weights going from the input window to the first hidden neuron. The thirteenth value corresponds to the bias (or threshold) of the first hidden neuron (refer Figure 2.4). Similarly, the next thirteen values are the weights going from the input window to the second hidden neuron followed by the bias of that neuron. For reasons known only to the original programmer (Professor Søren Brunak of CBS), the order of the input weights are reversed with respect to the actual order of nucleotides (meaning that the first four weights correspond to the last nucleotide in the window, while the last four weights correspond to the first nucleotide in the window). Finally, the next three values represent the two weights and the bias going from the hidden layer to the first output neuron, while the last three values are weights and bias for the second output neuron. Note that there is a positive discrimination in the weights attached to the third bit of the second input residue, in both the hidden neurons. That is the connection for the input i7 (refer Figure 2.4) for the first hidden neuron has a weight of 3.48135, and a weight of 2.46427 for the second neuron. With a window size of 3, the second input residue is the one in the middle of the window, and therefore the one being evaluated in the current window. In the event of it being a G residue, there would be ample firing from this position as the input bit would be 'on' or 1 for a G residue (encoded 0010). However, the synapse file does not specify any details of how the network is to analyse test data. All the extra information is hard coded into the predictor itself. Therefore when a user opens the URL for the CBS predictor, the synapse file is automatically read in and the specified network is built, and then the network is ready to accept the user provided test sequences.

2.2.9 Advantages and Disadvantages of Using Neural Network-predictors

The potential advantages and disadvantages of neural networks-based predictors over both statistical and normal computing analysis include:

Advantages

1. ANNs can represent any function, linear or not, simple or complicated.
2. The networks start processing input data without preconceived ideas. This allows for unbiased interpretation of the data, especially in unravelling subtle relationships between the various examples studied.
3. They are insensitive to moderate noise or unreliability in data.

4. Networks can be retrained using additional input variables not available at the time of first training. This property emphasizes their re-use and extensibility.
5. The value of the network's prediction is dependent on how representative the examples used in training were of the problem domain. As a result, a set of networks trained with different training suites can be used to provide complementing solutions for a particular problem.

Disadvantages

1. The artificial neural network is often considered to act as a 'Black box'. This means that although ANNs are good at delivering decent predictions, there are no explanations provided for the decisions. In theory one can examine the weights and gain some understanding of what is happening in the network (reference can be made here to the example synapse file– MySyn.txt discussed in Section 2.2.8). In practice, the list of weights is so long that it becomes impractical to examine them.
2. There are no set rules for network selection, implying that in building networks, there is no general theory specifying type of neural network, or network architecture, or learning algorithm that will work best for any given task. Thus, researchers rely solely on general rules based on observations made by previous network builders. The alternative is to try hundreds, even thousands of ANN topologies before the appropriate network is found to solve a particular problem [17].
3. Training networks is by no means trivial. It requires both time and expertise. As earlier indicated above, the choice of the training examples is very important, hence the trainer has to be very familiar with the ANNs' task domain.

Chapter 3

System Requirements Specifications for the NNI

In this chapter, the formal definition of the requirements specifications for the NNI are outlined. Furthermore, an assessment of the method used in extracting the requirements is given.

3.1 Introduction to the Problem Domain

The idea for this project came from Anders Gorm Pedersen at a meeting we (Joan Campbell-Tofte, and my supervisor, Peter Sestoft) held with him at the Center for Biological Sequence Analysis (CBS) of the Technical University of Denmark in January, 2002. At the meeting, Anders Gorm Pedersen expressed the need for a standalone application that should be able to simulate all of the CBS's neural network-based predictors. The proposed software will be referred to as Neural Network Interpreter (NNI). In drawing up the requirements specifications for the proposed application, I have used as foundation material the initial descriptions of the interpreter in a summary of the meeting made by Peter Sestoft. The document is attached as Appendix A. The functional and non functional requirements for the NNI were therefore drawn from this document, subsequent interviews between Anders Pedersen and me, as well as from discussions with Peter Sestoft. As the software should provide at least the same level of functionality as the CBS web-based predictors, I have also drawn inspiration from experimenting with the different prediction servers available at the CBS web site. The software requirements specifications were carried out using the document template suggested by Wiegers [26], and based on the 830-1998 IEEE Recommended Practice for System requirements specifications (SRS). The nonfunctional requirements are defined according to ISO 9126.

3.2 Product Scope

The NNI is intended to function as a standalone application with a graphical user interface whereby users can load one or more neural network-based predictors, load their test sequence in the form of DNA or protein sequences, and perform the neural network analysis on the user provided input data. The results of the analysis will then be presented to the user in graphical or in textual form. The input test data for the neural network-based predictors to be run in the proposed system will be exclusively deoxyribonucleic acid (DNA) or protein sequences.

3.3 Definitions of Acronyms and Abbreviations

OD: Prefix for business requirements or high-level objectives made by the client.

SR: Prefix for identifying users requirements or the specific behaviours the proposed software must exhibit [26].

3.4 Overview of the System Requirements

In the first part, the generalized requirements that will affect and more or less determine the nature of the proposed system are presented. This is followed by the specific functional and nonfunctional requirements for NNI.

3.4.1 Description of the Overall System

User Characteristics and Environment

The Neural Network Interpreter is expected to be used by biologists and chemists who are involved with biotechnology research and industry. They may have considerable information technology (IT) knowledge or they may not have IT expertise beyond the use of word processors. The use of the proposed software may typically be during one or both of the following phases in the research process.

Phase 1: A new project is about to begin. The researcher(s) use the predictors to analyse biological sequences extracted from the protein or nucleic acid databases in order to design appropriate study models for the enquiry they are about to initiate.

Phase 2: Well into the project or around the end when new sequence or experimental data have been generated, the predictors are used to analyse the data, as an aid to an understanding of the observations.

In both situations, the software will be used intensely for short periods with long time intervals between use. Hence even when users are quite familiar with the proposed software, it is not unlikely that they forget details of how to operate the system when next they have to use it. The maintenance of the system should be carried out by people with the knowledge of software engineering.

Product Perspective

OD-1: The Neural Network Interpreter will be a standalone application that should provide a framework for running neural network-based predictions. It is being built in association with CBS and will use synapse files of the neural network-based prediction systems in the format defined at CBS, many of which are currently in use at the CBS web site. One should thus obtain the same results whether an analysis is run on a predictor simulated by NNI or run on the equivalent prediction server available at the CBS website.

Product Functions

OD-2: The Neural Network Interpreter shall hold the information files for the different CBS predictors, as well as the synapse files for the neural network(s) which are part of the predictors. The information file which is unique for each predictor, shall detail its composition and mode of operation and is hereby termed the *parameter file*. So, there is a one to one correspondence between the predictor and the parameter file.

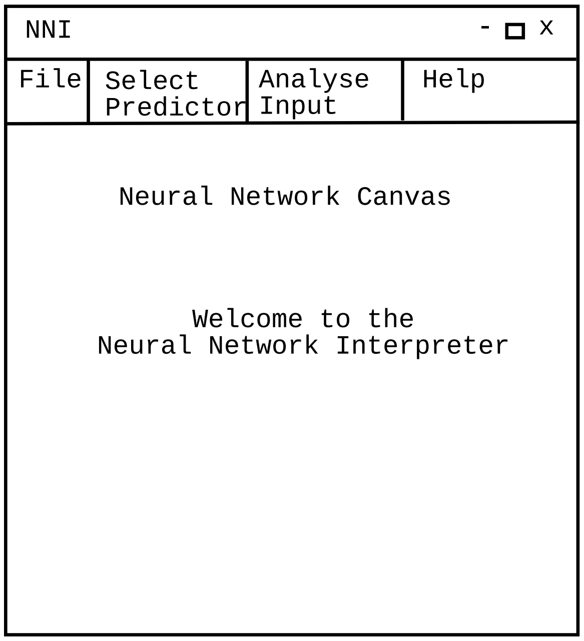

Figure 3.1: A sketch of a typical graphical user interface for the NNI

OD-3: The application shall provide a graphical user interface (GUI) as sketched in Figure 3.1 for the selection of the neural networks-based predictors, as well as for loading of user provided test data from files.

OD-4: The results of the analyses carried out in the Neural Network Interpreter shall also be presented to the user within the GUI.

OD-5: In summary, the Neural Network Interpreter should be able to simulate all the CBS neural network-backed predictors from the parameter files which it should carry, accept input test data submitted by users, drive the analysis of the data by the networks of the predictors and present users with reports and results of the analyses. The context diagram for the proposed application can thus be represented as shown in Figure 3.2.

System Constraints

OD-6: The Neural Network Interpreter should be operable on any operating system.

OD-7: The proposed application will be used to run only neural network-based predictions for biological data in the form of Deoxyribonucleic (DNA) and protein sequences.

Assumptions and Dependencies

OD-8: In order to ensure that NNI is platform independent (see SR-9 below), the application will be implemented using Java programming language. It is therefore assumed that every potential user of NNI has the Java virtual machine (JVM) installed on their computer. JVM is a platform-independent execution environment that converts Java bytecode into machine language and executes it. This is in contrast to

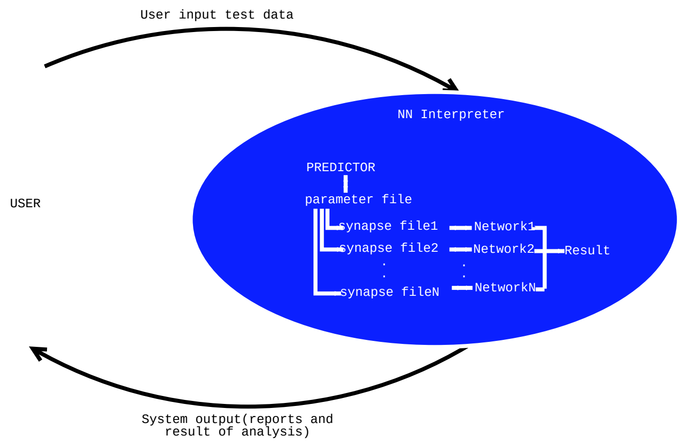

Figure 3.2: Context diagram for the proposed NNI. Users can load test input data. The system should drive the analysis of the input data, produce reports and results of the analysis.

the situation in several other programming languages that compile source code directly into machine code that is designed to run on a specific microprocessor architecture or operating system, such as Windows or UNIX.

3.4.2 Specific System Requirements

The functional and non-functional requirements for the Neural Network Interpreter (NNI) are presented in a logical sequence as follows:

Functional Requirements for NNI

SR-1: The Neural Network Interpreter shall provide GUI for the selection of neural network-based predictors and a programmable submission key.

Purpose: The NNI should support the running of any of CBS's neural network-based predictors. A GUI provides a convenient and easy stage for the selection.

Input: Users select the one or more predictor(s) that they are interested in. A selection can be deleted and a new choice made.

Processing: Handling of the selection should take no more than 5 seconds.

Output: If the selection is accepted by NNI, the user shall be presented with GUI asking for submission of the input test data. If the selection is not accepted, a message explaining the reason for failure and advice on how to correct the selection should be provided the user.

SR-2: In response to the user's selection, the application should read in the predictor's parameter file, constructing the associated neural networks in the process.

Purpose: Neural networks are trained to address specific problems and the synapse files specify the architecture of the neural network. Just as in the web-based prediction server (refer Section 2.2.8) the NNI should automatically build the networks in response to the user's selection.

Input, Processing, and Output: Once the user has selected a predictor, the NNI automatically accesses and processes the necessary data in the parameter file for the selected predictor. If the selection was valid, the user is presented with a confirmatory message and that the user could proceed with another selection, or submission of input data. Handling of this process should not take more than 5 seconds If it was invalid, and error message explaining the reason for failure and advice on how to correct the selection is provided.

SR-3: The Neural Network Interpreter should provide facility for submission and storage of user test data.

Purpose: Users need to submit their input test data to NNI.

Input: Input test data sequences.

Processing: Handling of the submitted test data should take no more than 5 seconds.

Output: If the input is accepted by NNI, the user shall be presented with the results of the analysis. If the input is not accepted, a message explaining the reason for failure and advice on how to correct the selection shall be provided the user.

SR-4: The test data shall be DNA or protein sequences and shall be submitted in the FASTA format [27].

Purpose: The NNI is being constructed to handle CBS-type networks trained to analyse DNA and protein sequences. The submission of test sequences in the FASTA format ensures that input test data is presented to the NNI in a standardized form. Aside from this convenience provided, users already familiar with the CBS predictors are used to submitting sequences in the format. Thus, the requirement SR-4 serves an added purpose, namely in giving the NNI some of the 'CBS web-predictor- feel'.

Input: Matched predictors(s) (as defined in SR-2) and input test sequences (as defined in SR-4)

Processing and Output are as in SR-3.

SR-5: The Neural Network Interpreter should be able to correctly analyse the submitted input sequences in the selected neural network-based predictor(s).

Purpose: As the networks in NNI are the same in the corresponding web-based predictors, the analysis of the same input data in the both systems should produce identical results.

Input: As in SR-4.

Processing: Handling of the sequence analysis should take no more than 5 seconds.

Output: When the analysis is completed, the user should be presented with the results of the analysis that have the same content as when the test input data used in this test is run on the web-based systems.

SR-6: The results of the analyses shall be presented to the user both in a textual and in a graphical format, with the possibility to save the text to file. In addition, the standard deviation of the results presented from that in the different networks shall also be calculated and presented in the textual form.

Purpose: Users need to see the results of the analyses. Where they are interesting, it will be useful to save them in the textual form.

Input : As in SR-4.

Processing: Handling of the analysis should take no more than 5 seconds.

Output: As in SR-5 and a file containing the result of the analysis in ordinary text form.

Non-functional Requirements for NNI

The nonfunctional requirements are defined using terminology from ISO 9126.

Table 3.1: Source for NNI's specific requirements. SR-1, SR-2, SR-3, SR-5, SR-9 were specified by Anders Gorm Pedersen; SR-6 was specified by Peter Sestoft; SR-4, SR-8 and SR-10 were specified by Joan Campbell-Tofte

Code for Functionalrequirement	Title for Functional requirement	Source
SR-1	Provision of GUI for selection of predictor(s)	AGP
SR-2	Automatic construction of selected predictor(s)	AGP
SR-3	Provision of GUI for loading user-provided input data	AGP
SR-4	Accept only DNA or protein sequences in FASTA format	JCT
SR-5	Analyse okayed input test data	AGP
SR-6	Presentation results of analysis to user in GUI	CBS web site and PS
SR-6	Present results of analysis to user in GUI	CBS and PS
SR-7	Provide a maximum response time of 1 min	JCT
SR-8	User friendliness	JCT
SR-9	NNI platform-independence	AGP
SR-10	NNI extensibility	JCT

SR-7: The System response time for using NNI starting from selection of predictor to result display by an experienced user shall not exceed 1–2 minutes.

SR-8: The system should be user friendly such that:

- 99% of users that are already familiar with using GUIs should be able to learn and successfully apply NNI within 10 minutes of using the system for the first time.
- User help notes giving important advice similar to those available on the web-based CBS predictors should be provided on NNI.
- Error messages shall describe the problem clearly and concisely, as well as provide constructive advice for recovering from the error.

SR-9: NNI should be very portable, that is it should be platform-independent.

SR-10: NNI should be extensible, meaning that it should be possible to incorporate and simulate newer CBS-like neural network-based predictors.

3.5 Traceability of the System Requirements

The source of the information used as basis for the specific system requirements are outlined in Table 3.1.

3.6 Assessment of the Method Used for Specifying NNI's Requirements

The use of a SRS document template helped me to properly understand the problem domain. This is because each subsection prompts the design analyst to make certain enquiries of the customer, some of which neither the software engineer nor the customer might have given a high priority during the early phases of the software development process. For example, the need to define the user characteristics and environment in Section 3.4.1 meant that I had to think of the context in which the proposed software could be used. Furthermore, the scheme made it easy to keep track of not only the functional requirements, but also of the nonfunctional design issues that also affect and determine the nature of the proposed software at an early stage. For example, considerations of system response time, provision of user help facilities, error information handling and command labelling, were addressed from the design phase.

Chapter 4

Design of the NNI

This chapter covers the reasoning behind the object oriented analysis and design steps that were used to model the NNI from the system requirements outlined in chapter 3.

4.1 Rationale for Using Object Oriented Principles for Software Development

Armed with a basic definition of the requirements specification, it was tempting to carry on the software development process using the traditional linear sequential or waterfall process model. However, a major requirement for the NNI is that it should be platform independent (refer specific system requirement SR-9). Hence the decision to use Java programming language as the implementation environment. As Java is an object oriented (OO) language with its well established $Java^{TM}$ Foundation classes (JFC) [18], it was more consistent to follow an OO paradigm for software development. The object oriented paradigm is characterised on the one hand by its emphasis on the creation of classes that hold both the data and the methods that are used in manipulating the data; and by encouraging the re-use of the classes across different applications and computer systems. Hence the NNI has been built from assembling newly designed and constructed NNI objects with the relevant JFC defined objects.

4.1.1 The Component-based Process Model

There are two distinct circumstances for OO design of software: one where the designer is starting from scratch to model a new product and the other situation, involving the modification of an existing product. With the former, a successful final product is more readily obtained by building prototypes and iterating through several versions with the customers [25]. As the proposed application is more of the second type, so that I was starting off with a pretty good idea of its requirements, I decided not to use exploratory prototyping.

Consequently, the software development process used in this project is the component-based process model described in Pressman [22]. Briefly, taking the requirements specification as foundation, the software development moved through an iterative process that started with modelling of scenario scripts or use cases which describe how the user (actor) would interact with the proposed NNI. Next, candidate classes needed to support the functionality of the system were extracted from the nouns that correspond to process entities or events in the use cases. As the objects encapsulate both the object's characteristics and behaviour, the responsibilities and collaborations of the classes were assigned by reference to the action verb phrases in the use cases. Initial class representations and assignment of responsibilities to the identified classes was carried out using class-responsibility-collaborator modelling (CRC) [20]. Finally, some of the dynamic behaviour of the system expressed as a function of specific events and time was modelled in both event trace and activity flow diagrams of the use cases.

As development of the application progressed through the implementation and test phases, some of the objects and operations which seemed necessary and valid earlier on were either revised, regrouped or removed altogether. As a result, the use cases and the associated workproducts presented in this report reflect the final model of the NNI.

4.2 Use Cases for the NNI

There are two main use case scenarios for the NNI.

Use case 1: User selects one or more predictors and uses these to analyse DNA or protein sequence data.

Use case 2: At any point during an NNI run, the user may access the NNI help function.

The use cases and their alternative courses are outlined below in Tables 4.1, 4.2 and 4.3. Each use case is cross-referenced to the system requirements which specify it.

4.3 NNI Architecture

The major classes designed for the NNI and their relationships are summarised in the UML class diagram presented in Figure 4.1. In the class diagram, the attributes and responsibilities of the classes are left out as they and the class collaborations are explained below and in Section 5.3. Essentially, the whole application is centered around the NNI's graphical user interface (NnIFrame) as the GUI provides all the means for user-interactions with the system. The filled arrows indicate that the user interface can be bi-directionally associated with several Parameterfile objects, one user-entered input TestData object, and several Result objects corresponding to the number of ParameterFile objects used. Note that a single ParameterFile object is in turn associated with several trained Network objects. Each of the Network objects is composed of several Neurons specified by input from a synapse file. The option dialog boxes used in the application to inform the user of the progress of the analysis or of errors represent subclasses of the major user interface. Hence the NNI message is connected to NnIFrame by an open headed arrow. Apart from the major NNI classes, a helper class Compute, was also designed to supply some of the methods required in the analyses of input data in NNI.

4.3.1 The ParameterFile class

As highlighted in Section 2.2.8, not all the information needed to run the network(s) for a given predictor is present in the synapse files. In providing the interpreter with the required additional information that is wired into the individual web-based prediction servers, the parameter file was introduced (refer Section 3.4.1). Like the synapse file which specifies a particular network, the parameter file is a text file. It specifies the properties and mode of operation for each predictor and therefore varies from predictor to predictor. Thus, the program must read the file and use the contents to construct a ParameterFile object. The data stored in the ParameterFile object can then be extracted when required and used to drive the simulation of that predictor which the text file describes. Consequently, the generalization required for the NNI is achieved by designing NNI to handle the different parameter files defined for the different predictors. In order to facilitate the reading of the parameter file by the system, the file names end in the suffix *param.txt* and the text file has been assigned a particular format as follows:

Table 4.1: Use case 1: Analyse input test data using one or more predictors. Based on functional system requirements SR–1 to SR–6

Actor:	User	
Description:	This use case describes the process whereby a user of the Neural Network Interpreter selects and runs one or more Predictors used in investigating protein phosphorylation on a test protein sequence. On completion, the user will be presented with the results of the analysis.	
Typical course of events:	**Actor Action**	**System Response**
	Step 1: The user starts the NNI by writing the command: java NeuNetInterpreter on the console.	*Step 2*: The window displayed welcomes the user to NNI and asks him/her to select desired predictor(s) from a drop-down menu option labelled Predictor.
	Step 3: The user selects and clicks on the menu item 'NetPhos-Serine' from the Predictor menu.	*Step 4*: The system displays the header for the predictor selected on the NNI canvas. The canvas also carries the message that the user may select another predictor or proceed by loading test data using the 'Load input' menu item on the menu 'Analysis'.
	Step 5: The user selects and clicks on another menu item marked 'NetPhos-Threonine'.	*Step 6*: The application displays a message listing that the 'NetPhos-Serine' and 'NetPhos-Threonine' predictors have been loaded. The user is also asked to select another predictor or proceed by loading test data.
	Step7: User selects and clicks on the menu item 'Load input' from the 'Analysis' menu. This brings up a filechooser, from which the user can select the test data containing protein sequences in FASTA format that he/she wishes to analyse. This interaction is completed when the button marked [open] is clicked on the filechooser.	*Step 8*: If the submitted input data is valid, the system encodes and presents the input data for analysis by the networks in the predictors. The results of the analyses from both predictors is presented to the user in the NNI result window in both the text and graphical forms, stacked one ontop of the other.

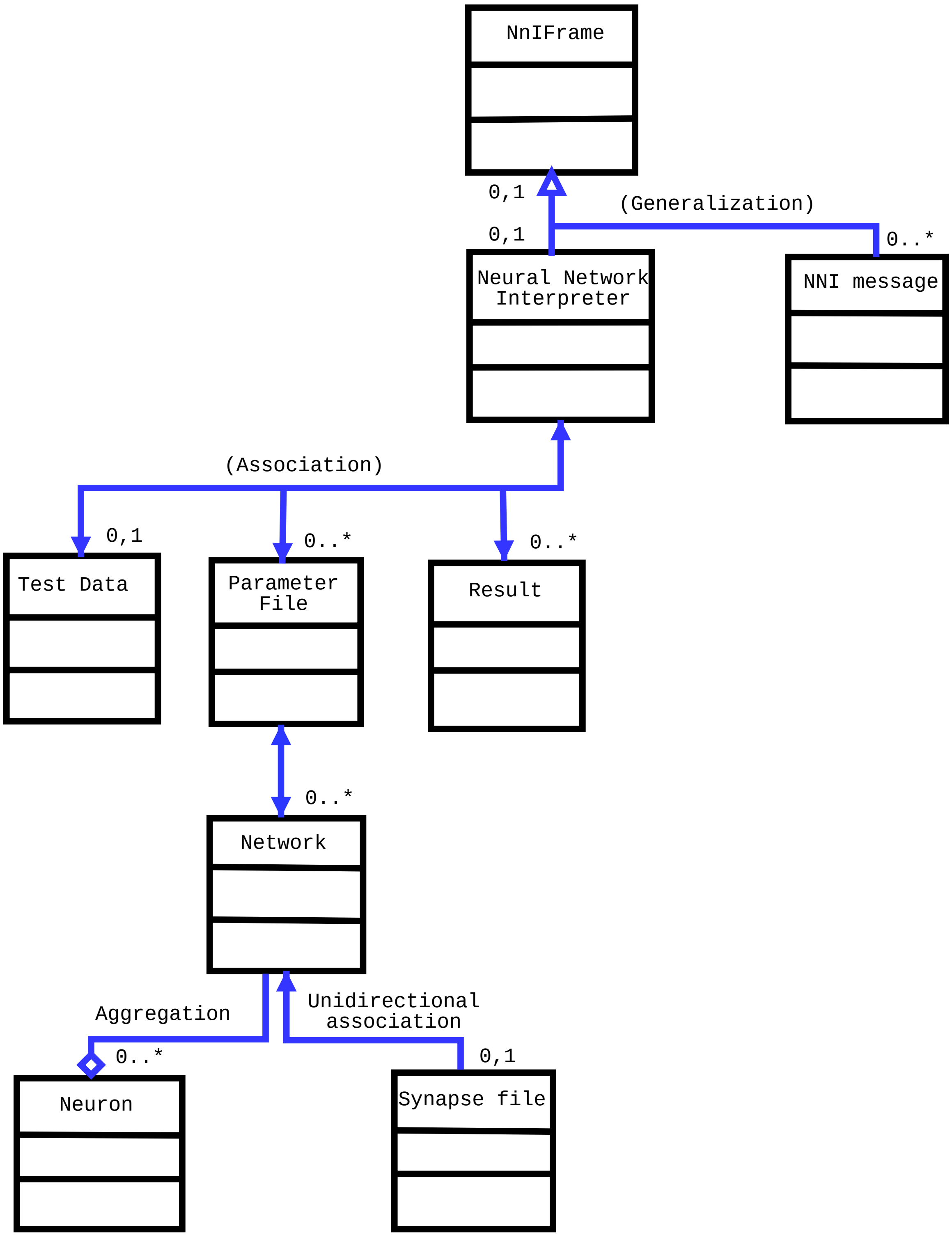

Figure 4.1: Class diagram for the major classes designed for the NNI

Table 4.2: NNI use case 1 continued

Alternative courses:	*Step 5*: The user has selected a second predictor designed to analyse different kind of data, e.g. selection of NetStart-dicots (used for predicting translation start sites in plant nucleotide sequences). This is not compatible with NetPhos-Serine and NetPhos-Threonine that are used to analyse protein sequences. The system displays a window with the error message 'Your recently selected Predictor is unacceptable. It has been loaded before or uses alphabet different from that in your previous selection'. Clicking [Ok] to acknowledge the message allows the user to select another predictor. Alternatively, the user may choose to reset the whole process or quit NNI. *Step 7*: The test sequence was not in the FASTA format. The system displays a window with the error message 'Wrong format for input data, resubmit'. Clicking [Ok] to acknowledge the message allows user to load another test sequence, restart or quit NNI. *Step 7a*: The test sequence is presented in the FASTA format, but the submitted sequences are composed of the alphabets for DNA. The system presents the user with an error message 'Wrong input sequences, check sequences and resubmit'. Clicking [Ok] to acknowledge the message allows user to reload another test sequence, restart or quit NNI.

Parameter File for the CBS Predictor: NetPhos-Serine

```
Predictor name: NetPhos-Serine
Output header: Prediction of serine phosphorylation sites
Allows use of unknown symbol: YES
Permits use of partial windows: YES
Synapse file(s): S_abcdSyn.txt S_bcdeSyn.txt S_cdeaSyn.txt S_deabSyn.txt S_eabcSyn.txt
Sequence pattern for window selection: S
Averages final predictor output: YES
```

Lines 1 and 2: The first two lines carry the name of the predictor and what it is used for, respectively. These are designed to be used for creating the predictor header when the results of the analysis are to be presented. Thus in the example, the predictor NetPhos-Serine is used for prediction of serine phosphorylation sites.

Line 3 Line three indicates whether the predictor's network(s) accept unknown residues or not. (Possible values here are 'YES' or 'NO'). Recall from Section 2.2.8 that each network has a defined input alphabet, meaning that it should only accept data containing characters from its alphabet. A YES value in line three implies that user submitted input data containing unknown (x) residues will not be rejected but will be correctly sparsely encoded as described earlier in Section 2.2.7.

Line 4 Line four specifies that following presentation of input data, the networks of this predictor will examine partially full windows as well as full ones. As mentioned earlier in Section 2.2.7, only the residue at the center of the moving window is evaluated by the neural networks. Therefore with networks that accept partially full windows, all of the residues in the test sequence get to be evaluated

Table 4.3: Use case : Using the NNI help function. Based on non–functional system requirement SR–8 (extending userfriendliness)

Actor:	User	
Description:	This use case describes the process of a user accessing the help function on ‘Fasta file format’.	
Typical course of events:	**Actor Action**	**System Response**
	The NNI window is displayed. Sequence analysis may or may not be running.	
	Step 1: The user selects and clicks on the menu item ‘Fasta format’ on the ‘Help’ menu.	*Step 2*: The system displays a dialog window with text: ‘Specify the input sequences intended for processing by browsing your computer and selecting a file in FASTA format. Example of two sequences in FASTA format: >seq1 ACGATGATCGATGCT- GAGCTAGCGCTCGCT >seq2 ASQKRPSQRHGSKY- LATASTMDHARHGFL Sequences can be submitted in upper or lower case. For more information about FASTA format, see ⟨http://www.ncbi.nlm.nih.gov/BLAST/fasta.html⟩. The user resumes the use of the system by clicking [OK].

as the ones at the ends can be placed at the center of the window when the sequence is padded with $(\text{window-size} - 1)/2$ blanks at both ends of the test string.

A 'NO' value at line four would imply that the network does not accept blanks and therefore begins to evaluate the input data at half window size counting from the first test sequence residue. Furthermore, this type of network would also stop analysis at the residue (window-size-1)/2 from the end of the sequence.

Line 5 Line five shows the list of synapse files which will be used for the construction of the predictor's networks.

Line 6 In certain predictors, only windows containing particular substrings beginning at the center window are evaluated. By contrast, all windows are evaluated in those predictors where a substring is not indicated in line 6. Hence, the letter S shown in line six of the example parameter file signalizes that in NetPhos-Serine, only serine (S) residues are evaluated for the possibility of being phosphorylation sites.

Line 7 Note that the input data will be evaluated independently by the different networks of the predictor. Therefore at the end of the analysis, the example predictor would have generated five sets of result, as it has five networks. The current practice with CBS prediction servers is that the final predictor output is obtained either by averaging the results from all the networks or by taking a juried decision of the results. In the latter, the result represented in the majority of the networks is considered to be the 'winner' or final result. Possible values in line 7 are therefore 'YES' for averaged results and 'NO' for juried results.

In summary, a ParameterFile object containing this information becomes the repository of the data needed to control the analysis in NNI. There should be one instance made for each activated predictor. A likely sequence of events when a predictor is selected in NNI would be that the program reads the parameter file, a ParameterFile object (with its associated Network objects) is created and stored in a data structure, an ArrayList that is placed within a field of the NnIFrame object. The Network objects are also stored in an ArrayList within the parent ParameterFile object. Placing the ParameterFile objects within NnIFrame ensues that the networks and the other data stored therein are handy for when the TestData object (see below) becomes available.

4.3.2 Input TestData Class

As implied in the Systems requirements (refer Section 3.4.2 SR–3), there shall only be one input test sequence per run during sequence analysis. For convenience of interaction with the networks (built within the ParameterFile object) the TestData object will also be stored within NnIFrame. The TestData object will have two string fields, one carrying the header extracted from the FASTA header, and the other will hold the input test data itself.

4.3.3 NnIFrame Class: the Graphical User Interface (GUI)

The center of activity in NNI is the GUI because this is where the user gets to initiate all his/her interactions with the application. The GUI therefore has the responsibilities of checking and storing selected ParameterFile objects, checking user submitted input data and loading input TestData objects, as well as providing the error and confirmation messages where necessary. To support these features, the Swing classes provided by the $Java^{TM}$ Foundation classes (JFC) for constructing graphical user interfaces will be used. Some of the packages from the older Abstract Window Toolkit package (AWT)– java.awt.* and java.awt.event.* which provide additional classes will also be used.

Generally, the design of the GUI will be kept simple, with the main functions provided as menus arranged in a logical sequence from left to right of a menu bar (refer sketch in Figure 3.1 of Section 3.4.1. The minor items will be extensions available as drop-down menu items from the menus. As the usage of the NNI is expected to be intermittent with perhaps long pauses (refer Section 3.4.1), the Swing option dialog windows will be presented to the user after each major interaction to either inform the user of the progress so far, or of any errors that might arise. On the whole, these will be kept to a minimum as it can get irritating to be confronted with too many dialog boxes.

For carrying out its responsibilities, the main NNI GUI has to collaborate with all the other NNI objects, specified by the ParameterFile, TestData, Result and Compute classes.

4.3.4 Result Class

CBS neural networks generally have two output neurons, implying that for each point along the input sequence there will be the need to hold 2 result values. Thus, the results of the analysis will be stored in a 2 dimensional array that will be stored in a field within the Result object. Again for convenience, the Result objects generated from each of the predictors selected will also be stored within an ArrayList field of NnIFrame, for ease of manipulation by the objects that should display the contents of the Result field..

Curiously, with the CBS predictors, it is only the data from the first of the 2 output neurons that is used for result presentation in both the averaged or juried predictor output. In accordance with OD-1 where it is stated that both systems should produce identical results, (refer Section 3.4.1) NNI remains loyal to that design.

4.3.5 Compute Class

In order to simplify the model, and reduce the number of responsibilities that would otherwise be placed on the ParameterFile object, a helper class named Compute was designed to hold some of the methods used in the analysis of input test data. Thus, the graphical user interface (NnIFrame), the predictor entities and input test data objects (ParameterFile, TestData, and Result objects) and the Compute class which encodes the interpretation logic of the system, are kept separate.

4.3.6 Data Storage in NNI

Both the ParameterFile and the eventual Result objects generated in an NNI session will be stored in ArrayLists. The ArrayList object has the ability to add, insert and store a collection of any objects on the fly. It is essentially a list implemented with an array and is provided for in the Java Collection (java.util.ArrayList). Like other Java collections, the ArrayList can be traversed by an iterator. The data structure has the special qualities of both being index-based (for easy access) and of providing dynamic resizing. As the number of predictors the user might want to use in any run cannot be predetermined, and the number of networks per predictor varies from one predictor to another, the use of an expanding list as data structure is well suited in NNI. Positional access takes constant time in the ArrayList. In addition, adding to or removing from the ArrayList has a time complexity of O(N-1), where N is the total number of elements in the ArrayList. Hence, adding or removing elements from the ends is very fast and has an amortized complexity approaching O(1) or constant time (for a long sequence of operations). However, iterating over elements in the ArrayList takes linear time or has a time complexity of O(N). As the steps involving the manipulations of the ArrayLists used in NNI will only involve adding elements at the ends, accessing elements via a specific index, and traversal can be achieved cheaply with an iterator [28], the list is perfectly suited to be used as the data structure type in NNI. As of now, the ArrayLists used in NNI never get to be particularly long as there are not that many predictors or networks per predictor (CBS currently has 18 neural network-backed predictors and of those I have seen, there are about 5 trained networks per predictor). Nevertheless, the ParameterFile object fields and the Network field do get accessed a number of times during the analysis of input data to justify the advantage provided by the fast element access in ArrayLists.

4.4 Activity Diagram for the NNI

While the UML class diagram models the static elements of the OO analysis and design systems, a dynamic behaviour of the proposed system can be represented in an event trace model or better still, with activity diagrams. The event trace model is used to show an ordered list of events between different objects typically in one use case. By contrast, the activity diagram models the flow of logic of the system. Both types of diagrams were used in the design process for NNI, but only the activity flow diagram is included in the report as it cuts across the logic of several use cases, it is used here to show the internal behaviour of the whole application pictorially. The activity flow diagrams also serve another purpose in that they can be used to depict ‘roadmaps’ through the system [29]. These routes or paths are invaluable during the design of integrated tests. The activity flow diagram presented in Figure 4.2 was also used in scoping the expected system behavior for the integrated test (refer Section 6.2). The UML activity diagram includes the following items:

- The filled circle depicting the starting point in the diagram.
- A solid bar called the synchronization bar. It shows that all the user interactions with the NNI start at the application’s GUI. This symbol is used to indicate potentially parallel processes. They can be performed in any order but in NNI, it is not necessarily such that all combinations of the possible steps are permitted. For example, it would be an invalid move to try to remove a predictor when one had not been previously selected.
- The rectangles represent processes or activities that are performed either by the user or by the system in response to stimulus from the user.
- The diamonds represent decision points within the application. Note that most of the decision boxes actually represent a set of nested decision boxes. They have been presented singly mainly because the activity diagram represents the system at a fairly high level of abstraction and also, in order not to clutter the diagram. For example, within the decision box following ‘Load TestData’, in Figure 4.2, the system first checks that at least one predictor has been loaded, if this is so, it checks if the submitted sequence is presented in the FASTA format. If the format is correct, NNI should also check if the alphabet in the test data matches that accepted by the Predictor. Thus with the one decision box, at least 3 different features are evaluated.
- The arrows model the flow order between the various activities.
- The text on the arrows represent conditions that must be fulfilled for the transition between the activities to proceed.
- All the activities end at a filled circle with a border. In the case where the last activity was invalid (for example, on invoking an error message), nothing further should happen in the system until the user initiates an alternative process.

.

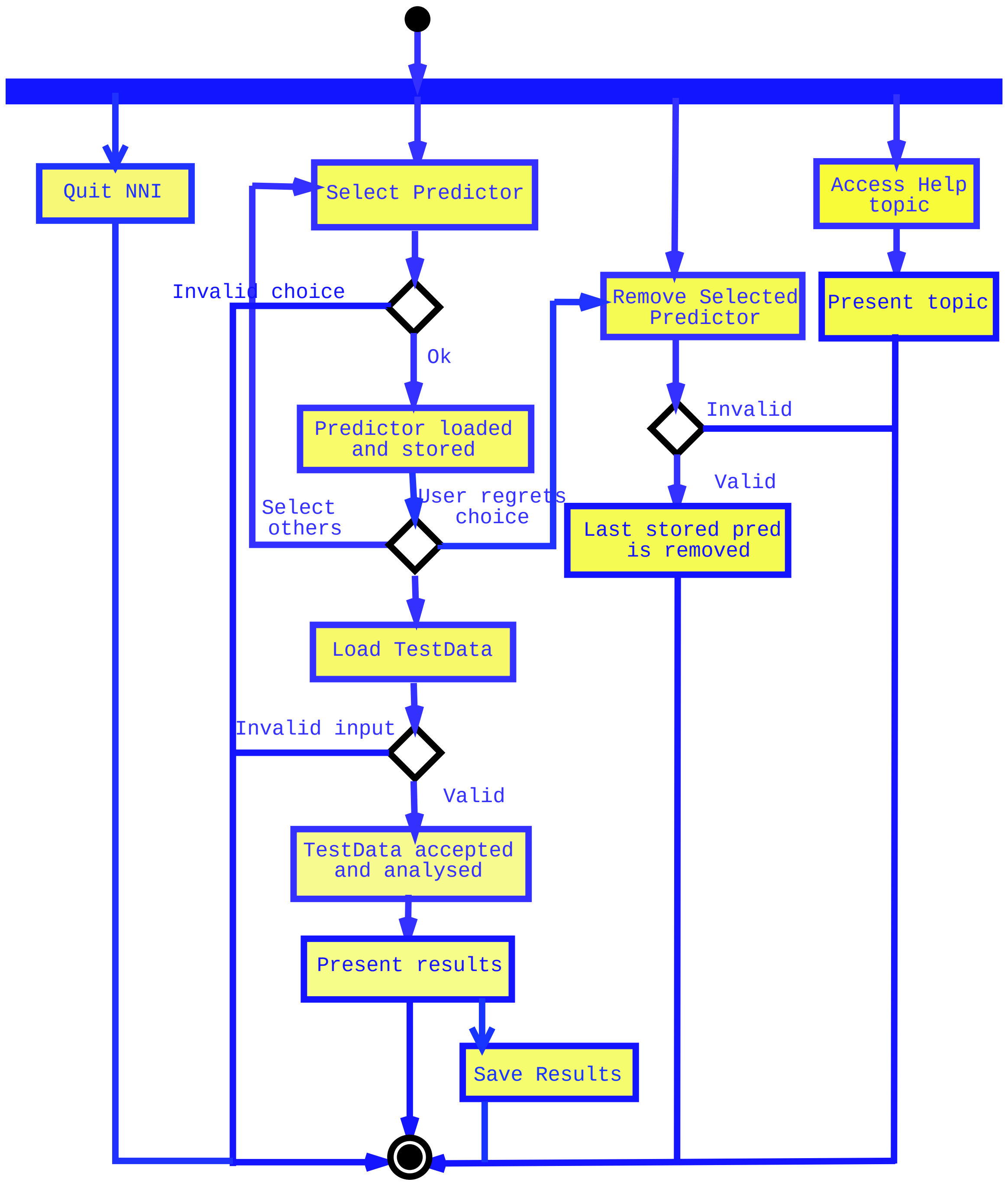

Figure 4.2: UML activity diagram for the NNI

4.5 Assessment of the Software Design Strategy

At the start of the thesis period, I was not particularly experienced with programming. However, during the course of the project, I have observed that with applying an iterative OO design process, I was better placed to systematically conceptualize and abstract the problem domain. In addition, it has been possible to access the degree to which the design alternatives fulfilled the system requirements. For example, after drawing an activity flow diagram for the NNI, it became obvious that quite a number of classes originally planned to be independent in the program could be combined to give a better design of the system.

Chapter 5

Description of the Implementation

In this chapter, the considerations that were made during implementation of the NNI will be treated. The motivation and rationale behind the design of system have been discussed in Chapter 4. First, the general guiding principles applied during source code construction will be mentioned and then the NNI features that required special attention will be discussed.

5.1 General Software Implementation Considerations

The source code for the NNI was written following the guidelines for the writing of secure Java code, as recommended by Eckel [28] and in ⟨http://java.sun.com/security/seccodeguide.html⟩. Briefly,

- As much as was possible, the use of static variables was avoided.
- Most object fields were declared private and accessed only via the appropriate getXXX() and setXXX() accessor methods.
- All input to the application are in simple text file format. Thus during file parsing, Java's input/output (IO) inbuilt security which demands that exceptions are implemented, was complied to. This ensured that the program sequence would break neatly if an error were to occur.

5.2 NNI-specific Implementation Considerations

As the NNI is intended to be faithful to the web-based CBS neural network-based predictors, a number of NNI facets (described below) had to be specially modelled and implemented. The NNI source code is organized into a directory named java, while the accompanying parameter and synapse files are packaged in the directory named user. For proper functioning of NNI, the subdirectories java and user have to be placed in the same directory.

5.2.1 Implementing the NNI GUI

The NnIFrame class (refer source code in Appendix B) is used to generate the major NNI GUI. The class is instantiated from the class NeuNetInterpreter (source code is in Appendix C). When the application is started, the NnIFrame constructor accesses the NNI subdirectory 'user' and generates an array of files from the files with names ending in the suffix *param.txt*, using the method listFiles() from the java.i.o package. These are the parameter files. Next, the ParameterFile constructor takes the contents of the list and generates an array **param** of equal size containing ParameterFile objects specified by the parameter files that were in the ‘user’ directory. The names of the predictors are then extracted using the accessor method of ParameterFile Class– getNameString() (refer Section 5.3.7), and passed onto the constructor for JMenuItem components as arguments in the build up of NnIFrame. Thus, the predictor names on the menu items of NnIFrame is generated dynamically and will always include all the parameter files in the ‘user’ subdirectory. The predictor names are then grouped into nucleic acid predictors or protein predictors for their respective submenus by first checking the predictors in the array **param**, using the ParameterFile method getPredictorType().

NnIFrame represents a typical application control framework built by extending the Swing top-level container, JFrame. The control framework is a particular type of application constructed to respond to events. The functionality in the NNI GUI is provided by a JMenu bar and its associated menu items. Each menu item is in turn handled by the different ActionListener classes that are attached to them. The organization of the different components and the ActionListener classes that are used in the NnIFrame class is show in Table 5.1. The rest of the NNI main window holds a canvas made up of a Swing intermediate container, JPanel. The canvas is managed by a CardLayout manager which displays one of 3 JPanels depending on which menu item is active. For example, it displays the welcomePanel carrying the welcome message and start-up information when the user opens NNI. During predictor selection, the names of the selected predictions are displayed on a displayPanelsLoadedPanel, and at the end of a successful run, the resultsPanel is used to display the results both in the textual and in the graphical format.

5.2.2 Input Data Presentation in the NNI

As indicated in Section 2.2.8, the order of the input weights for the neurons in the hidden layer of CBS networks are reversed with respect to the actual order of input signals. This indicates that in a network having an input window of three residues A, C and G (each encoded as four binary digits represented by positions 1, 2, 3, and 4), the weights for a hidden neuron are in the following order:

$$G_{v1}, G_{v2}, G_{v3}, G_{v4}, C_{v1}, C_{v2}, C_{v3}, C_{v4}, A_{v1}, A_{v2}, A_{v3}, A_{v4} \tag{5.1}$$

In NNI, it was more convenient to adopt a uniform pattern for reading in the weights for the neurons in all layers, namely that they are read in without reversing. Therefore the weights equivalent to those described in the sequence marked 5.1, are actually in the order:

$$A_{v1}, A_{v2}, A_{v3}, A_{v4}, C_{v1}, C_{v2}, C_{v3}, C_{v4}, G_{v1}, G_{v2}, G_{v3}, G_{v4} \tag{5.2}$$

The strategy that was then adopted in NNI to compensate for the reversed hidden neuron weights that is built into all the CBS synapse files, was to reverse the input test data. Depending on whether the networks accepted or did not accept partial windowfuls of input data, the new string was sparsely encoded after padding or no padding, respectively. The resulting binary numbers were stored in an array. At the start of the sequence analysis (refer the next Section 5.2.3), windowfuls of the encoded data were extracted and presented as input to the networks. In so doing, for the input string ACG whose weights have been placed in the loaded neural network as illustrated in the order 5.1 above, sparse encoding the reversed string GCA into binary numbers using the code (from Section 2.2.7):

$A = 1000, C = 0100, G = 0010, T = 0001$

would result in an input of:

0010	0100	1000

Thus the NNI inputs get to be correctly matched with the corresponding weights. The program segment presented below shows exactly how this feature of sparsely encoding input test data was implemented within the method encodeTestData() from the class Compute.

Table 5.1: Component and ActionListener classes of the Class NnIFrame

Menu	Menu items	ActionListener class	Responsibilities
File	Reset NNI	Resets	Resets the system by clearing all objects stored from the previous run.
	Quit NNI	Quits	Exit NNI
Predictor	Select nucleic predictors	GetParam	Provides constructor which takes a parameter file name as argument and builds a ParameterFile object
	Select protein predictors	GetParam	Same as with Select nucleic predictors above
	Remove most recently selected predictor	MinusPredictor	Removes the last ParameterFile object within the field **pffiles** in NnIFrame
Analysis	Load data	LoadData	Invokes the TestData constructor.
	Save Results	SaveData	Saves results of analysis to file.
Help	The Basics	Basics	Provides general information on using NNI.
	Fasta format	Fasta	Provides information on using the FASTA format.
	About NNI	About	Provides information about the making of NNI.

```
//Method for sparse encoding test data into N binary digits.
    public static byte[] encodeTData (String testd, ParameterFile pfile){
        //First extract necessary  info from first network in pfile.
        Network firstnet = pfile.getfirstNetwork();
        //System.out.println("Within encodeTestData() & network is extracted");
        int numalph = firstnet.getN();
        String st = pfile.getAlphabet();
        int win = firstnet.getWindow();
        String input = revContent(testd);
        byte[] coded;
        int startfill;
        //Calculate the size for the array to hold all the encoded test data.
        if (!pfile.getWindowPartial()){
            int bytesize = input.length()*numalph;
            coded = new byte[bytesize];
            startfill = 0;
        } else {
        //The array to hold encoded test data is larger by windowSize-1.
            int bytesize = (input.length() + (win-1))*numalph;
            coded = new byte [bytesize];
            startfill = ((win-1)/2)*numalph;
        }
        for (int i=0; i<input.length(); i++){
            char c1 = input.charAt(i);
            int num = getIndex(c1, st);
            for (int j=0; j<numalph; j++){
                coded[startfill] = (j==num ? (byte) 1:0);
                startfill++;
        }
        }
        return coded;
     }
```

Briefly, the method takes the test string sequence (extracted from the TestData object) and a ParameterFile object as arguments. Using information extracted from the ParameterFile object on the input alphabet and nature of window content accepted by the networks, the input test sequence is encoded into an array of bits that is either as large as input string length multiplied by the total number of input alphabets for the networks, or this table padded by [network window-size - 1]/2 times the size of the alphabet on either side of the input string.

5.2.3 Generalizing NNI– Using the Information Stored in the ParameterFile Class

The ParameterFile class holds all the information required to set up and run the predictors in its fields (refer source code in Appendix D). The blueprint for the ParameterFile object is provided by the parameter file which is taken as argument to a ParameterFile constructor invoked within the Actionlistener GetParam class of the NnIFrame class. Since the parameter file also contains the list of the predictor's networks, they (the networks) in turn get constructed by calls to the Network class constructor (see source code in Appendix E), as the ParameterFile object itself is been built. This way, soon after the user has selected the required predictor, the networks are installed and the stage is set for sequence analysis.

The implementation of the actual routine for sequence analysis within NNI was complicated by the many possible combinations of the conditions stipulated in the parameter files for the different predictors. For example, while NetPhos-Threonine (a predictor for threonine phosphorylation sites in proteins) accepts both partially full windows and input data with the unknown symbol X, NetStart-vertebrates (a predictor for start sites in DNA sequences) accepts partially full windows, but not input data with the unknown symbol X. On the other extreme, is the test predictor JoanPF which neither accepts partially full windows nor test data containing the unknown symbol. In order to better illustrate the reasoning behind the source code for the implementation of the analysis of data in the NNI, the pseudocode for the whole process from when NNI is activated to when the Result objects are constructed can be summarised as follows:

1. When NNI is started, the NnIFrame constructor is called from NeuNetInterpreter.java and the NNI GUI carrying the predictor names on menu items is built. Refer Section 5.2.1 for details of the construction of NnIFrame.

2. The user selects predictors and these get loaded and stored in the ArrayList field– **pffiles** of the NnIFrame object.

3. Following the loading of a valid input test data, the method analyseData() specified by the Result class is invoked on the input sequence and a ParameterFile object extracted from the ArrayList **pffiles**. It is within this method that the method encodeTestData() from the class Compute is invoked. The encodeData() method generates an array of bits from the test string. Refer Section 5.2.2 for details of how the array of bits is generated.

4. Next, there is a check to determine if the networks accept partially full windows. If the value is true, the input string is padded, while if the value is false, there is no padding.

5. While keeping tab of which residue from the original test string is at the middle of the current window, a check is made to find out if the predictor has a search substring. If there is a substring, the presentation of data proceeds with a windowful at a time, but only that data where the substring occurs at the middle of the window is evaluated. The result is then stored in a 2-dimensional array of doubles know as **netResult**, at an index corresponding to the position of the substring on the input string. If the value for substring was an empty string, then the data from all input windows are evaluated and the result are stored as described above for the predictors which only evaluate windows with particular substrings. All the **netResult** objects generated in the different networks are then stored in an ArrayList field– **totResult** of NnIFrame.

6. Finally, the method setNetOutput() specified in the class Result is used to calculate the overall results of the predictor from the values in **totResult**. The manner in which the final predictor output is calculated depends on whether the information extracted from the ParameterFile object is true or not for the ParameterFile field **averagesResults**. If the value is true, the results are averaged. If the **averagesResults** field is false, then the overall predictor output is that which is represented in most of the individual networks' results.

5.2.4 Provision of Explanatory Notes for the Predictors

One of the key attractive features of the web pages carrying the CBS prediction servers is the use of concise explanatory notes. These inform the user of what the prediction server can do, how to present the input data, as well as how to interpret the results. In an attempt to maintain the 'same CBS prediction server feel' within NNI, some notes have been provided in the Help menu of NNI. However, because of the need to be concise in NNI, it was not possible to make NNI help notes quite as explicit as those on the CBS web sites.

5.2.5 Implementing NNI Extensibility

Day by day, researchers come up with new questions that neural networks are being trained to investigate. Therefore, it would be useful to extend the range of predictors NNI should handle. Note that this mode of generating the NNI GUI ensures that the GUI will always carry the currently available predictors. Thus, extensibility is built into the system from the start. So, as long as the parameter file and the synapse files are written in the current CBS format and placed in the folder indicated within the message invoked with the menu item marked ’The Basics’ on the Help menu, the new predictors will be automatically loaded with the old ones when the application is started. Similarly, when predictors become obsolete because newer and better tools are available, a biologist without the need for IT competence beyond the use of word processors can just access the NNI subdirectory user and remove the disused files.

5.3 Implementation of the NNI Classes

The major classes that were built for the NNI include: NeuNetInterpreter, NnIFrame, ParameterFile, TestData, Result, Network and Neuron. In addition, the Compute class was constructed to provide some service methods, while several inner classes were implemented as ActionListener classes or simply helper classes to support the many activities on NnIFrame (refer Table 5.1 in Section 5.2.1). The NNI classes and the relationships between them have been present in Figure 4.1, Section 4.3.

5.3.1 Class NeuNetInterpreter

Refer source code in appendix C. The class NeuNetInterpreter provides the main() method to the whole application. Thus this is the starting point for the NNI.

5.3.2 Class NnIFrame

Refer source code in appendix B. The class NnIFrame extends the javax.swing.JFrame class and is responsible for constructing the NNI GUI. The rationale behind its design has been extensively discussed in Section 4.3.3, while its implementation is described in Section 5.2.1. In addition to the many Swing component classes used, the NnIFrame inner classes include:

Class EmptyCanvas: extends JPanel and is used for setting the preferred size for NnIFrame.

Class MyFileFilter: implements FileFilter of java.i.o package and is used to extract parameter files during the construction of NnIFrame.

Class DisplayPanelsLoaded: extends JPanel and is used to Display the selected predictor names. When a new predictor is added or deleted, the panel is refreshed to reflect the currently loaded predictors.

Class MinusPredictor: used to handle removal of the last predictor in the ArrayList pffiles field of NnIFrame.

Class Resets: Provides reset function in NNI.

Class Quits: Shuts down the system.

Class GetParam: used to invoke the ParameterFile constructor on selection of predictor names.

Class Basics: Activates Help menu item on general NNI information.

Class About: See Table 5.1.

Class Fasta: See Table 5.1.

Class LoadData: See Table 5.1. Invokes the TestData constructor when the user submits valid input data.

Class ShowResults extends JPanel and is used to package and present the results of the analysis. The ShowResults class also specifies the getTextResults() that collects the results of the analysis in textual form and presents it to the an object of the class SaveData which then saves the data to file.

Class SaveData: See Table 5.1. The SaveData object is used to save the results of sequence analysis to file.

Class Welcome: extends JPanel and is the first canvas to be shown when the system is started.

Class Display: extends JPanel and takes a 2-dimensional array as argument and generates the graphical representation of the results. Objects of the class Display are invoked within the ShowResults class during presentaion of the results of the analysis.

5.3.3 Class Network

This class specifies the ANNs. See source code in Appendix E. The public methods are the various accessor methods such as getWindow(), getN(), getOutputalph() and getInputalph(). In addition there are several methods that are used by the Network constructor during parsing of the synapse file. These include, skipToken(), getDouble() and getNumber(). The class also contains the important method computeOutput() which is used to calculate the output from each layer of the network.

Class Neuron

The class Neuron is implemented as an inner class of Network, as they are both closely associated. The Neuron class has fields for the weights and bias values that are used in the network's analysis. In addition, the Neuron has the very important method, calOutput() that is used for calculating the output from the individual neurons. It is indeed this method that is called iteratively on the neurons of the constituent network within the other important method computeOutput() of the Network class.

5.3.4 Class NnIException

The class NnIException is the Exception class that was designed for NNI. See Appendix H for its source code.

5.3.5 Class Result

The Result class was designed mainly to store the results of the analysis in its 2-dimensional array field. In addition, it specifies the method analyseData(), used for submitting window size test data to the network and for controlling the whole sequence of events, most of which is described above in Section 5.2.3. The class has also been assigned the method setOutput() for calculating the final average or juried predictor output. In order to facilitate the display of the results, the Class is also provided with an accessor method, getNetOutput(). Refer Appendix I for the source code for the Result class.

5.3.6 Class TestData

The TestData class constructor is invoked to build the TestData object which stores the input data when a valid input data is submitted to NNI. As explained earlier, it is stored within NnIFrame, so that it can be accessed easily during analysis. It has two string fields for storing the sequence title and the test string, respectively. In addition, TestData is furnished with two accessor methods, getHeading() and getInd() to extract the data within the fields. Refer Appendix G for the source code.

5.3.7 Class ParameterFile

See Appendix D for the ParameterFile source code. The rational behind the design of the ParameterFile class has been discussed in Section 4.3.1 and the application of the information stored in the object's fields has been outlined in Section 5.2.3. The ParameterFile fields include:

Three fields of type String, 'nameString', 'head', 'sub' corresponding to the predictor name, predictor name and what it is used to predict, and the substring that is searched for during the analysis of sequence data, respectively. Note that **sub** can be an empty string, implying that all of the data in the input string gets evaluated.

'netwk' of type ArrayList is used for storing the Network objects that are associated with the predictor specified in the ParameterFile object.

Three fields of type boolean, 'acceptsX', 'windowPartial', 'averagesResult' . True value with **'acceptsX'** indicates that the networks will accept input data containing the unknown residue X. With **'window-Partial'**, a true value implies that input data can contain blanks, while **'averagesResult'** is true if the overall predictor output is averaged over the results from all the constituent networks. A false value here would mean that the result is determined by a juried decision over all the results from the networks in the predictor.

The ParameterFile class also specifies the public accessor methods, getNameString(), getHead(), getWin-dowPartial(), getSub(), getNetObject(), getfirstNetwork(), getX(), getAlphabet() and getAveragesResult(). In addition, there is the method getPredictorType() used in grouping the predictors during the construction of NnIFrame (see Section 5.2.1). The getPredictorType() method provides its function by extracting the input alphabet from the first network and checking whether it is DNA or not using the method isDNA() from the class Compute. If isDNA() returns a true value, the predictor gets grouped as a nucleic predictor, otherwise it is grouped as a protein predictor.

5.3.8 Class Compute

The Compute class is the helper class that provides the general methods used by the different classes. Refer Appendix F for source code. The methods include:

- getUsedCode(): Used in comparing the contents of the alphabets in two strings. A true value indicates that both strings contain the same alphabet.
- isDNA(): Used in testing if a string contains characters typical of DNA.
- getIndex(): Used in the encodeTestData() method for assigning bits to characters.
- revContent(): Used in reversing the test data before encoding.
- extract(): Used for extracting binary digits for presentation to the network.

- encodeTestData() has been discussed in Section 5.2.2 above.

Chapter 6

Evaluation of the Neural Network Interpreter

This chapter gives an overview of the test of the Neural Network Interpreter. The test protocol included unit, integrated and validation tests.

During software development, errors are introduced from various sources. For example, there can be errors due to insufficient understanding of the problem domain, or defects introduced from typographical errors, logical errors or errors due to poor programming practice. Errors are also introduced during restructuring of the program. By testing the application, one can execute the system's functions with the purpose of uncovering and subsequently correcting any defects before product release. There are several partially automated software tools that can be used to ease the process of designing and writing tests. An example is JUnit that is freely available from ⟨www.junit.org⟩. Nevertheless, for a smaller application like the NNI and for me to get properly acquainted with constructing test suites, I decided I would get the most out of the project by constructing the test suites and carrying them out myself.

As indicated in earlier chapters, the construction of NNI started with the systematic definition and analysis of the system requirements and design models. As these work products contain the same semantic models (classes, attributes and functions) that are used at the implementation step, reviews carried out earlier on in the project could uncover errors in design and prevent their propagation to the next development phases. However, due to the time limitations for this project, we did not have formal review processes at both the requirement specification or design phases. At the implementation phase, the testing of the Neural Network Interpreter was carried out at three levels. Firstly, unit tests that are designed to ensure that each class in the application is constructed and behaves as it should were carried out. In the later stages of the implementation, when several classes were joined together to build more complex objects or carry out subsystem functions, integrated tests were carried out. Finally, at completion of implementation, software validation or functional tests (Blackbox tests) based on the requirement specifications were also carried out.

6.1 Unit Tests

During the implementation of each of the main NNI classes (outlined in chapter 4), a main() method was put into each class. The resulting code was then used as a test bed for the class when the programs were run. In this way, all the class attributes could be viewed and some class methods were exercised using embedded 'System.out.println()'-statements. Examples of such unit tests carried out for NnIFrame.java (that codes for the NNI user interface), Network.java (class which specifies the backpropagation trained network) and TData.java (class which is used to create and store the test data for NNI analysis), are outlined in Tables 6.1,

Table 6.1: Unit test 1: for NnIFrame class

Input	Classes, fields and function exercised	Expected Results	Actual Results
Type: java NnIFrame at the console	class: NnIFrame; fields:NnIFrame menu bar, menus, menu items and WelcomePanel	An early version of the NnI GUI with a menu bar, menus, menu items, and panel displaying the welcome message should appear on the screen.	As expected.

Table 6.2: Unit test 2: for Network class

Input	Classes, fields and function exercised	Expected Results	Actual Results
Type: java Network at the console	classes: Network and Neuron; fields: all Network's and Neuron's fields; Methods: Network's constructor (3), Network.getN(), Network.getOutalph()	So far, so good. alphabet for network is ACGT output alphabet for network is i. nI= 12 nH= 2 n0= 2 Window size is 3 2 so far. 4 at the end. 4 i. Hallo test network	As expected

6.2, and 6.3, respectively. The source code used in the unit tests are presented in Appendices J.1, J.2, and J.3, respectively. The input data used in the test sequences are presented in Appendices J.4 (Testseq1.txt) and J.5 (seq2.txt).

6.2 Integrated Tests

The integrated tests served to check how the classes behaved in their new environments, as well as to test those methods that could not be tested individually within their parent object, because they depend on other objects to function. Ideally, the integrated tests should be based on structural tests (sometimes referred to as white-box or internal tests). This implies that the integrated test suite must include enough input test data to ensure that all the methods in the constituent classes are called; both true and false possibilities in the if statements and all branches of switch statements are executed; and that the lower, middle and upper iterations for all loops are tested. Even for a smaller application like NNI, it easily becomes quite complex to design and run such tests. Therefore, the basis path testing technique of McCabe [21] was used to derive

Table 6.3: Unit test 3: for TestData class

Input	Classes, fields and function exercised	Expected Results	Actual Results
Type: java TestData Testseq1.txt at console	class: TestData fields: TestData.heading and TestData.ind; functions: TestData's constructor (3), TestData.getHeading(), TestData.getInd()	This is within testData constructor 3 This heading is: >testseq 1 The test string itself is: agatc tgagagagagagagag tagatgatagatagcgctag Entered getHeading >testseq 1 agatctgagagag agagagagtagatg atagatagcgctag	As expected
Run input TestData.java Testseq3.txt from console	Same as above	Testseq3 is not in FASTA format. Error message is displayed on JDialog box stating that the format is wrong and 'Wrong text format' appears on the console.	As expected.

a logical complexity measure for the execution paths in the Neural Network Interpreter. The value defines the number of linearly independent paths running through the program. An independent path is any path moving from start to a finishing point in the program that introduces at least one new set of processing statements or condition. The number of independent paths provides an upper bound for the number of tests that must be conducted to ensure that all the program statements are executed at least once. The resulting paths are then used as guides for defining the integrated tests. Briefly, an activity flow chart depicting the NNI control structure in the user-initiated activities was drawn from the use cases for the NNI as described by Pressman [22] and [29] (Refer Figure 4.2 in Section 4.4). In order to assign the minimum execution paths for the NNI, the activity flow chart was further mapped into a corresponding acyclic flow graph shown in Figure 6.1. Each node in the flowgraph represents one or more system statements, while the edges represent the control flow. The edges must terminate at a node, even if the node only represents an error message. The nodes that contain a condition are referred to as predicate nodes and are characterised by having two or more edges emanating from them. As indicated earlier in Section 4.4, the activity flow diagram represents a model of the system at a very high level of abstraction. Therefore the paths are derived by abstracting away from the detailed decisions. For example, the many different conditions present in the program execution steps between the storage of the input test data object (node 8, Figure 6.1) and the result presentation (node 9), were not included amongst the paths used for designing the integrated tests. Note that for the purpose of optimizing the extraction of the minimum execution paths, the missing decision points could have been included in the charts by extracting them from the relevant source code. However, due to the time limitation for this project, it was not possible to go into as much detail in designing the integrated tests as would have been desirable.

The integrated tests suite (outlined in Tables 6.4 and 6.5), is designed along the paths extracted from the flowgraph. Wherever an integrated test and a validation test had identical input data, then reference was made to the validation test.

6.3 Black-box or Validation Tests for NNI

In defining the system requirements, (Chapter 3) both the higher-level business requirements and specific system requirements were identified (Refer Section 3.4.1 for overall system's description OD1–8; and and Section 3.4.2 for requirements SR-1–10). The black-box tests suite or validation tests outlined below in Tables 6.6, 6.7 and 6.8 are based primarily on the specific user requirements specifications, since they also express the points raised in the more generalized business requirements. The test number, classes exercised, and the relevant requirements specification are indicated with each test entry. Testseq3, OrdFASTAtext.txt, and TranSparam.txt, TranSyn.txt refer to the input data files used in the SR-4 and SR-10 tests. They are presented in Appendices J.6, J.7, J.8 and J.9, respectively.

6.4 Overall Assessment of NNI Performance During Testing

Although there are a number of the integrated tests that could have been carried out for the system that were not done due to time limitation, the NNI has performed well in the unit tests and the integrated tests that have been carried out so far. However, in the validation tests, NNI failed in providing a calculation of the standard deviation of the overall results from the predictor. This can be corrected by adding a few lines of code within the method setOutput() of the class Result, for calculating the standard deviation of the results in the results from the different networks. In addition, the NNI has not been tested in a MacOS environment. Plans are underway to do so as soon as possible. Even when all the tests have been executed, it is impossible to rule out the possibility that there are some errors in the program that have not found and corrected. I have

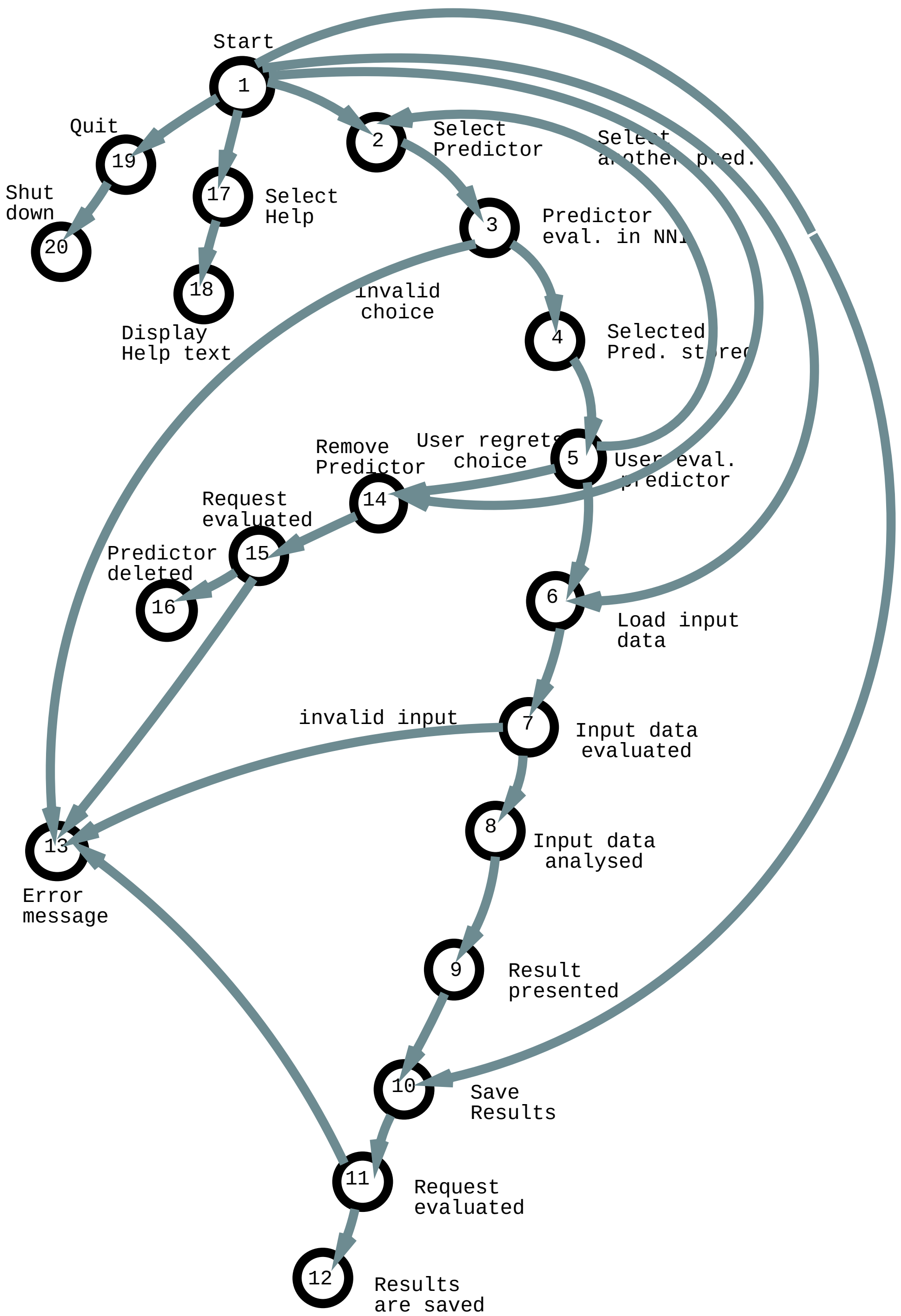

Figure 6.1: Flow graph derived from the NNI use cases

Table 6.4: Integrated tests for NNI. *The steps enclosed in the brackets represent repeated steps.

Path	Input data	Classes exercised	Expected Results	Actual Results
path 1: 1-(2-3-4-5)*-6-7-8-9-10-11-12	This represents a valid test run. To be tested in black-box tests			
path 2: 1-(2-3-4-5)*-2-3-13	User has selected NetPhos-Serine and NetPhos-Tyrosine. In the third round, user clicks on NetStart-2	GetParam in NnIFrame	Error message– 'Your newly selected predictor is unacceptable. It has been loaded before or used alphabet different from that previously loaded'.	As expected
path 3: 1-(2-3-4-5)*-14-15-16	User has selected NetPhos-Serine and NetPhos-Tyrosine, regrets last choice and activates the 'Remove most recently selected predictor'	GetParam and MinusPredictor in NnIFrame	NetPhos-Tyrosine is removed from the display panel	As expected
path 4: 1-(2-3-4-5)*-6-7-13	User submits invalid input data, for example Test-seq3.txt (refer Appendix J.7) which is not in FASTA format	LoadData in NnIFrame	Error message– 'Wrong format for input data. Please resubmit data in the FASTA format'.	As expected
path 5: 1-14-15-13	User activates 'Remove most recently selected predictor' without previously selecting one	MinusPredictor in NnIFrame	Error message– 'No predictor to remove'.	As expected
path 6: 1-10-11-13	User activates 'Save Results' without having run an analysis	SaveData in NnIFrame	Error message– 'Sorry no results to be saved yet'.	As expected

Table 6.5: Integrated tests for NNI continued

Path	Input data	Classes exercised	Expected Results	Actual Results
path 7: 1-6-7-13	User activates ‘load input’ without having first loaded a predictor.	LoadData in NnIFrame	Error message– ‘Sorry you cannot load input data without having loaded a Predictor.’	As expected
path 8: 1-17-18	User activates the ‘Fasta format’ menu item on the ‘Help’ menu	Fasta in NnIFrame	Dialog box appears with message on FASTA format.	As expected
path 9: 1-19-20	User activates ‘Quit’ menu item on ‘File’ menu.	Quits in NnIFrame	NNI shuts down.	As expected

tried to systematize the tests and found it a very interesting and rewarding exercise. A usability test designed to find out how user friendly the software is could also have been interesting and useful. The usability tests were however left out due to time constraints.

As at the time of writing this report, only four CBS neural network-based predictors have been passed through test runs in NNI. Again due to limitations in time, it was not possible to test NNI against more CBS predictors. For example, we have not tested a predictor which calculates its final output by jury. Hence it is crucial to test NNI against all the neural network-based CBS predictors. This is important as it would increase the confidence that NNI can do all that the web-based prediction servers can do and more.

Table 6.6: Validation tests for NNI

System Requirement	Input data	Classes exercised	Expected Results	Actual Results
SR-1: Provision of GUI for selection of predictors	Start NNI by typing java NeuNetInterpreter at the console.	NeuNetInterpreter, NnIFrame and ParameterFile	NNI GUI with a canvas and menu items for predictors	As expected
SR-2: In response to user's prompting, construct predictor with its networks.	Select the predictors NetPhos-Tyrosine and NetPhos-Threonine. In order to test their presence, load valid data, e.g. seq2.txt (refer Appendix J.7)	GetParam in NnIFrame, ParameterFile	Result of the analysis in textual and graphical form	As expected
SR-3: Provide facility for submission of test input data.	Select predictor NetPhos-Serine and activate the menu item 'Load input' in the 'Analysis' menu	LoadData in NnIFrame TestData	A file chooser window appears for browsing and selection of input file.	As expected
SR-4: Test data shall only be DNA or protein sequences in FASTA format	Select a predictor, for example NetPhos-Serine and on activating 'Load input', submit Testseq3.txt.	LoadData in NnIFrame and TestData	Error message– 'Wrong format for input data. Please resubmit data in the FASTA format.'	As expected
SR-5: NNI shall correctly analyse valid input test sequence	Input is same as in SR-2 test	NnIFrame, ParameterFile, TestData, Network, Result, Compute, NnIException	Same as in SR-2 test	As expected

Table 6.7: Validation tests for NNI (cont.)

System Requirement	Input data	Classes exercised	Expected Results	Actual Results
SR-6: NNI shall present the results of the analysis in both the textual and graphical formats, show standard deviation amongst the networks results and also give user the possibility to save the results	Same as in SR-2 test.	Result, ShowResult and SaveData in NnIFrame	Presentation of correct results, standard deviation and possibility of saving the results.	The results can be saved and are correctly presented in the textual and graphical format but there is no standard deviation value.
SR-7: Provide maximum response time from start to result presentation of 1-2 minutes, when used by an experienced user.	Same as in SR-2 tests, except that System.out.println(“The time is:” + System.currentTimeMillis()) were included at the beginning of the program in NeuNetInterpreter.java and at the end after ShowResults class has been invoked on NnIFrame.java.	All NNI classes	Expected results are that Stoptime minus starttime will be between 1 and 2 minutes	36 seconds
SR-8: Provide user friendly notes to explain some details in using NNI	Activate ‘The Basics’ on the ‘Help’ menu.	Basic in NnIFrame	An option dialog window with some helpful hints	As expected

Table 6.8: Validation tests for NNI (cont.)

System Requirement	Input data	Classes exercised	Expected Results	Actual Results
SR-9: NNI should be portable, that is the software should be operable and produce same results in any operative system.	Start NNI by typing java NeuNetInterpreter at the console in Windows or Linux environments that have a JVM. Select a predictor JoanPF, load valid data Testseq1.txt	All NNI classes	Running NNI in both systems should lead to the presentation of results of the analysis in the textual and graphical format.	As expected. Note that NNI could not be tested in MacOS environment because I did not have a Mac machine with the latest JVM that is needed to run the javax.Swing classes.
SR-10: NNI extensibility	Open the subdirectory 'User' and save the files TranSparam.txt and TranSyn.txt (refer Appendix J.8 and J.9, respectively). TranSparam.txt specifies a fictitious newly acquired predictor for predicting membrane proteins in signal transduction and TranSyn.txt is the synapse file for the only network in TranSparam.txt. Start NNI by typing java NeuNetInterpreter at the console.	NeuNetInterpreter, NnIFrame and ParameterFile	NnIFrame with TranS appearing as one of the protein predictors.	As expected

Chapter 7

User Manual for the Neural Network Interpreter

In this chapter, a short general description of the NNI is followed by instructions for installing and using the software.

7.1 Introduction to Using the Neural Network Interpreter (NNI)

NNI is a standalone biological sequence analysis tool that is designed for simulation of the CBS neural network-backed predictors. That is, with NNI installed on a local computer, one is able to use predictors with the same capabilities as the neural network-based CBS prediction servers available at ⟨http://www.cbs.dtu.dk/services/⟩. Running the software on any PC however requires that one has the Java plug-in (the Java virtual machine), loaded on the local machine and that NNI is downloaded and installed. This user manual includes a step by step guide on how to use the software, as well as a trouble shooting guide detailing the reasons and solutions for the NNI error messages you may encounter.

7.2 Installing NNI and Getting Started

7.2.1 Installing the Java Plug-in

The Java plug-in needed for running NNI can be downloaded free of charge from the web site ⟨http://java.sun.com/getjava/download.html⟩. Click on the download button and you do not have to do anything more as the downloading process you have activated will do the rest. According to the accompanying Sun feature article, the entire process, from click to completion, can take as little as five minutes, depending on your network connection.

7.2.2 Installing and Setting up NNI

NNI files are package in two folders named Java and user. The Java folder contains the Java Class files whereas the parameter files and the synapse files are located in the user folder. You have to have both folders in the same directory.

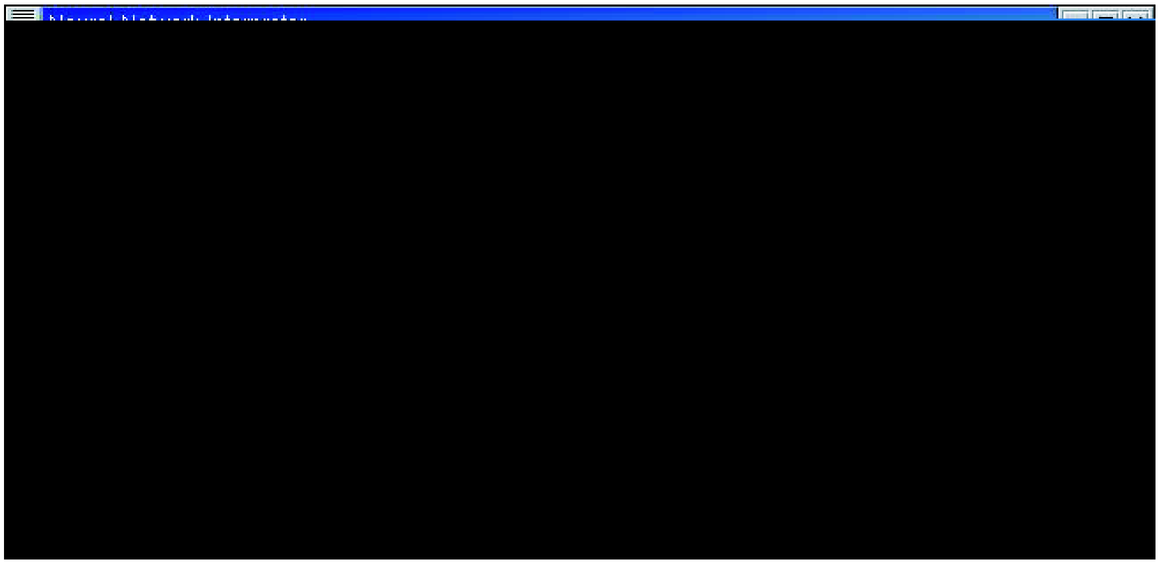

Figure 7.1: Graphical user interface of the NNI

7.3 Running NNI

You start NNI by typing: ‘java NeuNetInterpreter’ on a prompt from the console, ensuring that you have the correct path to the program. This brings up the NNI welcome window shown in Figure 7.1.

7.3.1 Composition of the NNI Window

The typical NNI window is made up of:

- A closable frame with the resize, close and iconifying buttons.
- The NNI menu bar with menus and menu items described below in Section 7.3.2.
- The NNI canvas carries most of the NNI messages. The information displayed on the NNI canvas varies from window to window. For example, the welcome page carries the welcome message, while the results are displayed on the same canvas (refer Figure 7.2) when these become available at the end of the sequence analysis.

In addition to the main window, NNI also uses small dialog windows (refer Figure 7.3) to provide users with necessary pieces of information or with error messages.

7.3.2 NNI Components and Keys

File menu The File menu has the menu items:

- ‘Reset NNI’, provides the restart function for clearing the system typically at the end of an analysis, but it works just as well at any other point.
- ‘Quit NNI’. Only activate this menu item if you wish to quit NNI as activating it will cause the system to shut down.

Predictor menu The Predictor menu is composed of 3 submenus, namely:

- ‘Select nucleic predictors’ carries menu items with the names of predictors used in analysing nucleic acid sequences.
- ‘Select protein predictors’ carries menu items with names of predictors used with protein sequence data.
- ‘Remove most recently selected predictor’: Activating this menu item will delete the most recently selected predictor. You can use it repeatedly to get at the other selections, but if you wish to remove all the predictors previously selected, it is faster to use the ‘Reset’ menu item.

Thus, all the predictors currently available from NNI are partitioned between the ’Select nucleic predictors’ and the ’Select protein predictors’ menus depending on whether they are used for analysing nucleic acid or protein sequences, respectively.

Analysis The Analysis menu carries the menu items:

- ‘Load input’ for loading the user provided input sequences.
- ‘Save results’ used for saving the results of the analysis in text form to a file.

Help The Help menu provides three menu items labelled:

- ‘The Basics’ is a link to extra information about the different predictors, and advice on how best to use them in NNI.
- ‘Fasta format’ is a link to information on how input data should be formatted.
- ‘About NNI’. Activating ’About NNI’ provides an information dialog box about the creation of the software.

7.3.3 Operating the NNI

There are 2 main ways of using NNI. These include:

- Using one or more related predictors to analyse an input test data.
- Accessing the Help facility on NNI. For example, clicking on the ‘The Basics’ menu item in the Help menu.

Running One or More Related Predictors on Input Data

1. At the NNI welcome window, select a predictor you wish to use. You can select more predictors so long as they are of the same type. For example, you cannot select several nucleic predictore and one protein predictor in a single run of NNI. You will be presented with error messages if you are trying to make invalid choices. If in doubt of which predictors to select, you can refer to the CBS web page at ⟨http://www.cbs.dtu.dk/services/⟩.
2. When you have finished with selecting predictors, you can load your input test data by using the file chooser dialog box that comes up in response to your clicking the ‘Load input’ menu item reachable from the menu ‘Analysis’. Note that unlike the CBS web prediction servers, NNI does not provide the possibility of copy-pasting your input sequences. They have to be stored to a file and loaded therefrom. With the aid of the file chooser, browse to the directory where you have saved your sequences. The input data is loaded when you click the button ‘open’ on the dialog box. As with the CBS web applications, the sequences have to be submitted in the FASTA format.

```
Example of 2 files in FASTA format:
>seq1
AAACCTTGGCGTAGACAGATATAGGCAG
>seq2
ASQKRPSQRHGSKYLATASTMDHARHGFLPRHRDTGILDSIGRFFG
NMYKDSHHPARTAHYGSLPQKSHGRTQDENPVVHFFKNIVTPRTPP
KSAHKGFKGVDAQGTLSKIFKLGGRDSRSGSPMARRELVI
```

3. At the submission of a valid input test data, the analysis is set off and the results of the completed analysis can be viewed on the NNI result canvas. See 7.2 for an example.

4. You can save the textual form of the results to file by clicking on the ‘Save results’ menu item reachable from the menu ’Analysis’. This will bring up a file chooser dialog box. Write the name for the file in which you wish to store your results and select the directory you wish to keep it in, then click on [Save]. In Windows, you get best results by ending the file name in ‘.doc’.

Adding a New Predictor to NNI

1. To add one or more new predictor(s) to the NNI, you need to download and save the parameter file and the accompanying synapse files in the NNI subdirectory, ‘user’. There should be as many file names ending in the suffix param.txt as you have new predictors. That is there should be one file with a name ending in param.txt for every one of the new predictors.

2. Check too that you have synapse files for all the networks that are listed in the the parameter files. They should end in the suffix ‘Syn.txt’. They too are to be installed within ‘user’.

3. Start NNI and the names of the new predictors should appear in the appropriate group of predictorsif the proper files were placed within ‘user’. If not, you will get an error message like ‘Could not find the file for ‘XXname’ for input’. In which case, you should check with the person you got the parameter files from that you have received all the synapse files that should come with the parameter files.

7.3.4 Troubleshooting

Errors occurring during the use of NNI will cause it to show an error message and stall. The typical NNI error messages look like the dialog window shown in Figure 7.3.

The software can be set moving once the cause of the error is identified and removed. In this section, the typical error messages from NNI are summerized in Table 7.1. In addition, the causes and what can be done to restore the system to working condition are indicated.

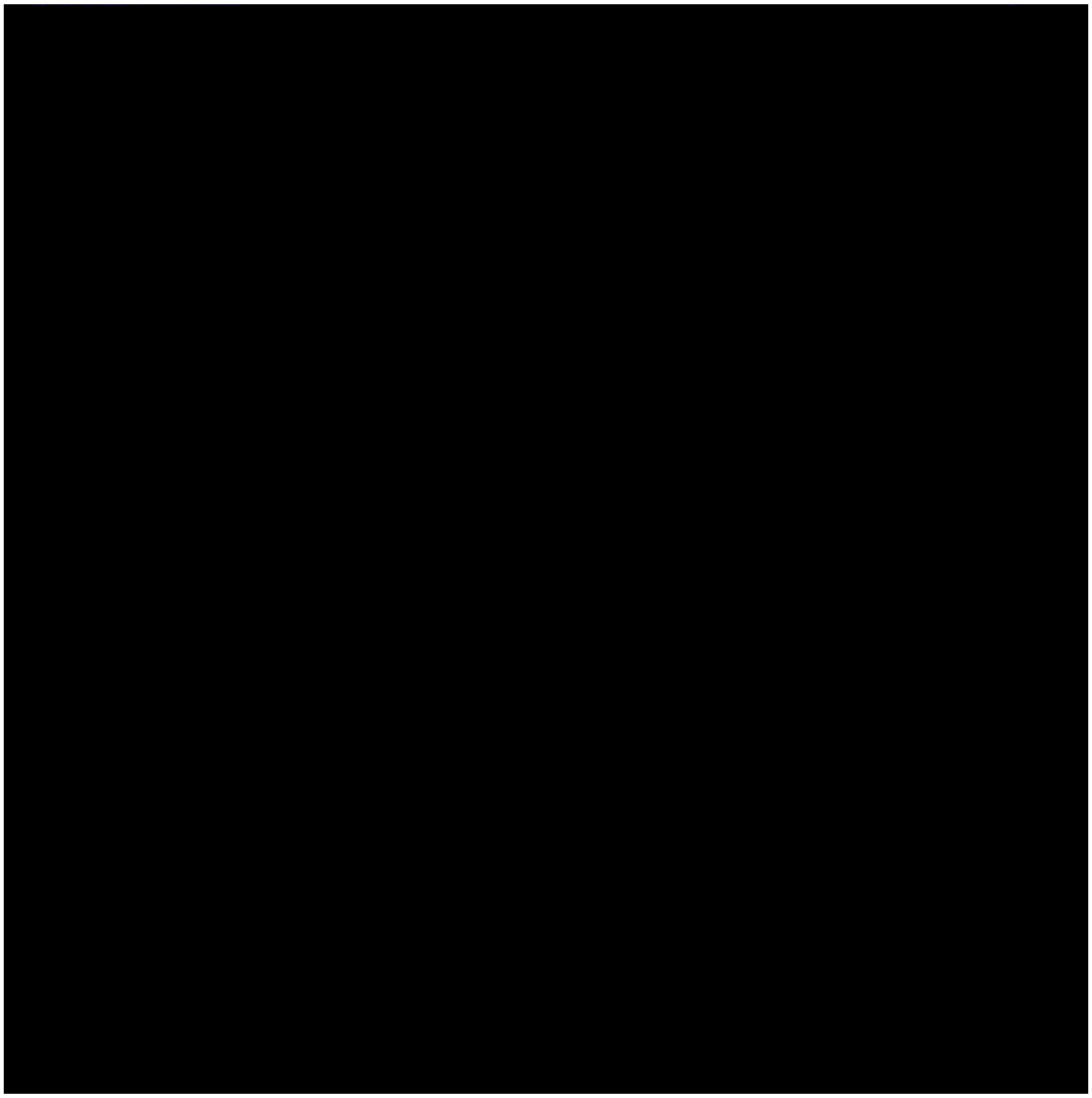

Figure 7.2: Sample output from NNI after analysis of DNA sequence Testseq1 using predictors JoanPF and Netstart-vertebrates.

Figure 7.3: Typical error message presented in NNI

Table 7.1: Error messages in NNI

Error message	Cause	Solution
‘Could not open file ‘NetGene2’ for input’	User has selected a predictor whose associated synapse file(s) are not placed within the NNI subdirectory ’user’.	Check that the files are placed correctly or complain to your predictor supplier.
‘No predictor to remove’	User has activated ‘Remove most recently selected predictor’ without previously selecting one.	Comply with the instructions.
‘Wrong formt for input data. Please resubmit in FASTA format’	User has tried to load input data that was not in FASTA format	Comply.
‘Alphabet of input sequence conflicts with the alphabet of the predictor’	User has tried to load non-matching input data with selected predictors	Comply.
‘You cannot select predictors once input data is loaded. Consider File → Reset NNI’	In the middle of analysing valid input data, user tries to select a new predictor	Comply.
‘Your newly selected predictor is unacceptable. It has been loaded before of uses alphabet different from that previously loaded.’	See message	Comply.

Chapter 8

Conclusions

This thesis has involved the design and implementation of platform independent standalone application which should be capable of simulating any of the CBS neural network based predictors. In this chapter, a short summary of the project will be presented and my own evaluation of the resulting work products and process will be given.

8.1 Summary of the Project Process

In developing the conceptual model for the NNI, as well as during the implementation process, I have drawn a lot from the IT courses I took at the IT university of Copenhagen in the earlier semesters. The courses– Software development, Design of data structure and user interface, and Object oriented programming have been particularly relevant. The same goes for the Bioinformatic course I took at CBS, since this is where I first heard about artificial neural networks. The project started with elucidation of the systems requirement and then design. So, in spite of coming from a weak programming background, once the design was in place, the reiterations of implementation, testing and redesign proceeded smoothly with excellent direction from my supervisor– Peter Sestoft.

8.2 Evaluation of the Product

The major work product from this thesis is the Neural Network Interpreter. It is implemented in Java language which makes it portable as all it needs to be run on any platform is a freely available Java plug-in– the Java Virtual Machine. It is small (1580 lines of code) and fast, as it takes approximately 36 seconds from when the application is opened to when the results of the analysis is presented on the screen. It has the potential of providing all the functionality in the 18 CBS web based artificial neural network-backed predictors. It satisfies most of all the functional requirements and has performed well in all the tests so far carried out. A usability test still needs to be carried out though, to ensure that it is user-friendly. In order to achieve the generalization that was required of the NNI, the concept of a parameter file was introduced. The parameter file has been designed to contain all the possible variants of predictor properties characteristic of the CBS predictors. Therefore, by storing the predictor variables in a ParameterFile object and constructing the NNI to analyse data driven by information extracted from the objects, it was possible to achieve the generalization desired for the NNI. All the input to the software is in the form to text files. This is very convenient as the parameter files and synapse files could be stored as flat databases. There is of course the downside that the important data in the files could be overwritten. This is why the files in the NNI subdirectory ‘user’ will be made readable but not writable.

8.3 Evaluation of the Process

The greatest motivation for me in choosing this project was to have the experience of building software that should have practical application from scratch. With the NNI, it has been possible for me to take a trip into software development. The trip has been particularly interesting because it has meant that I could have stops, albeit small ones sometimes, in the landscape of gathering information about likely projects, extracting the requirements specifications, modeling the proposed system and finally implementing and testing it.

In using the object oriented paradigm of software development, I came to appreciate its strength when it became necessary to change the design about half way into the implementation. It turned out that because with OO, one is all the time dealing with objects, it is not so complicated to move the make changes to a design. Such changes can be considered as shuffling the objects. Thus object responsibilities change and collaborations are adjusted to fulfil the new model. In a particular example with NNI, the original idea was to make the Network class the central piece and what became known as the ParameterFile object, the worker or helper object. According to this line of thought, the GUI should have carried networks, not predictors as it does now. Once we started considering different predictors, it soon became obvious that while some predictors have only one network others have several. Then there were all the different ways data is presented, e.t.c. Thus, it became more relevant that the ParameterFile object, should be the central piece. After a few days of looking again at the design, it was actually quite easy to shuffle the objects around and build NNI on the ParameterFile object. This gave a simpler and tidier model. For example, NNI extensibility is achieved by simply adding on a parameter file for the new predictor to the NNI subdirectory, ‘user’.

8.4 Future work with Interpreter

The NNI is extensible. It may be necessary in the future to remove some of the predictors that are part of NNI now, when they become obsolete. The parameter files are loaded onto the NNI GUI from an internal file list. Hence, it is easy for a researcher to open the NNI subdirectory ‘user’ and delete the obsolete parameter files and accompanying synapse files. More urgently, NNI should be tested against **all** the CBS prediction servers. This will not only boost confidence in NNI’s soundness, but only then can we truly declare that NNI can replace the CBS web predictors.

Appendix A

Preliminary Description of the Proposed Neural Network Interpreter from a Meeting held on January 9, 2002

Participants: Peter Sestoft, Anders Gorm Pedersen and Joan Campbell-Tofte.

Original Danish text Designe og implementere fortolker for synapsefiler for neurale netværk med grafisk brugergrænseflade. Kan bruges til at indlæse og evaluere en sekvens og tegne en profil af nettets output hen over sekvensen. Normalt har man fem netværk (og altså fem synapsefiler) og foretager en afstemning eller tager gennemsnit af deres output. Kunne også vise ‘usikkerhed’, spredningen af de fem netværks output. Undersøg om Biojava har relevante klasser til indlæsning af sekvenser.

English translation Design and implement an interpreter for synapse files (neural network), supported by a graphical user interface. The proposed software should be used to simulate the neural network, read, analyze input data as well as present the results of the analysis. Normally, there are five networks (i.e. 5 synapse files) involved in the analysis and the final results are computed by either making a juried decision or taking an average of the results from individual networks. The output from the proposed software could also indicate the deviation among the results from the different networks. Find out if Biojava has Classes defined to assist in reading sequence data.

Appendix B

File NnIFrame.java

```
/*This program defines the NnIFrame class v2.5 28-10-2002,
 *  after latest clean-up.
 * The GUI consists of a closable frame, a menu bar,
 * a canvas and all the actionListener
 * classes required to access it. Also contains ’Add new Predictor’, and the
 * different reset and remove functions. */

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import java.awt.geom.*;

//Code for the basic NnIFrame GUI
public class NnIFrame extends JFrame  {

    // td holds input data to selected predictors, once input data has
    // been loaded.
    private TestData td;
    // pffiles contains selected ParameterFiles
    private ArrayList pffiles = new ArrayList();
    // allResults contains Result objects of analyses
    private ArrayList allResults = new ArrayList();
    //Array to hold all the ParameterFile objects
    private ParameterFile[] param;

    //Components of NnIFrame
    private JMenuBar mb = new JMenuBar();
    //menus for the NnIFrame menubar.
    private JMenu
        filemenu =      new JMenu("File"),
        predictormenu = new JMenu ("Predictor"),
        nucleicmenu =   new JMenu ("Select nucleic predictors..."),
        proteinmenu =   new JMenu ("Select protein predictors..."),
        loadmenu =      new JMenu("Analysis"),
        helpmenu =      new JMenu("Help");

    /* cardlayout is used for showing welcomePanel, resultsPanel,
     * addPanel, confirmPanel and displayPanelsLoadedPanel alternately
     * in the same area of the screen during execution.*/
    CardLayout cardlayout =                        new CardLayout();
    JPanel card =                                  new JPanel();
    Welcome welcomePanel =                         new Welcome();
    ShowResults resultsPanel =                     new ShowResults();
    DisplayPanelsLoaded displayPanelsLoadedPanel = new DisplayPanelsLoaded();

    // directory where parameter files, synnapse files reside.
    // System.out.println("user directory is: " + user);
    File java = new File(".").getAbsoluteFile().getParentFile();
    //  System.out.println("java directory is: " + java);
    File nni  = java.getParentFile();
    //  System.out.println("NNI directory is: " + nni);
    File user = new File(nni, "user");
    //  System.out.println("user directory is: " + user);
    File thesis = new File(nni, "thesis");
    //  System.out.println("thesis directory is: " + thesis);
    File dir  = new File("user");

    NnIFrame(){
        super("Neural Network Interpreter");
        //Read in info on predictors from the dir "user"

        File[] filesInDir = user.listFiles(new MyFileFilter());
        ParameterFile [] param = new ParameterFile [filesInDir.length];
        try {
            for (int i=0;i<filesInDir.length;i++)  {
                System.out.println(filesInDir[i].getName());
                param[i] =  new ParameterFile(user, filesInDir[i].getName());
                System.out.println(param[i].getNameString());
            }

        } catch (NnIException x) {
            System.out.println("Failed to create parameter files");
        } catch (IOException y) {
            System.out.println("Failed to create parameter files");
        }

        //Compose menuItems for 'File'
```

```
JMenuItem[] forfile = {
    new JMenuItem ("Reset NNI",
                   KeyEvent.VK_R),
    new JMenuItem ("Quit NNI", KeyEvent.VK_Q)};
forfile[0].addActionListener(new Resets());
forfile[1].addActionListener(new Quits());
for (int i = 0; i < forfile.length; i++)
    filemenu.add(forfile[i]);
mb.add(filemenu);//adding filemenu 1

//Composing menuItems for ’Predictor’. First get sizes for
//min[] and mip[] by running check for predictor type in param
int countNucleicPred = 0;
for(int j = 0; j<param.length; j++)
    if (param[j].getPredictorType()) countNucleicPred ++;

JMenuItem[] min = new JMenuItem[countNucleicPred];
JMenuItem[] mip = new JMenuItem[param.length-countNucleicPred];

int nucl = 0; int prot = 0;
for (int k = 0; k<param.length; k++){
    if (param[k].getPredictorType()){
        min[nucl] = new JMenuItem(param[k].getNameString());
        min[nucl].addActionListener(
                   new GetParam(param[k].getNameString()+"param.txt"));
        System.out.println("The string on GetParam is "
                           +(param[k].getNameString()+"param.txt"));
        nucl++;
    } else {
        mip[prot] = new JMenuItem(param[k].getNameString());
        mip[prot].addActionListener(
                   new GetParam(param[k].getNameString()+"param.txt"));
        System.out.println("The string on GetParam is "
                           +(param[k].getNameString()+"param.txt"));
        prot++;
    }
}

for (int n = 0; n < min.length; n++){
    nucleicmenu.add(min[n]);
}
predictormenu.add(nucleicmenu);

predictormenu.addSeparator();

for (int p = 0; p < mip.length; p++){
    proteinmenu.add(mip[p]);
    }
    predictormenu.add(proteinmenu);

    predictormenu.addSeparator();

    JMenuItem removePredmenu =
        new JMenuItem("Remove most recently selected predictor");
    removePredmenu.addActionListener(new MinusPredictor());
    predictormenu.add(removePredmenu);
    mb.add(predictormenu);//adding predictormenu 2

    //MenuItems for 'load ...'
            JMenuItem load = new JMenuItem("Load input");
            load.addActionListener(new LoadData());
            loadmenu.add(load); // loadmenu.addSeparator();
            JMenuItem save = new JMenuItem("Save results");
            save.addActionListener(new SaveData());
            loadmenu.add(save);
            mb.add(loadmenu); //adding loadmenu 3
    //MenuItems for 'Basics'
            JMenuItem basics = new JMenuItem("The Basics");
            basics.addActionListener(new Basics());
            helpmenu.add(basics);
    //MenuItems for 'Fasta file format'
            JMenuItem fasta = new JMenuItem("Fasta format");
            fasta.addActionListener(new Fasta());
            helpmenu.add(fasta);
    //MenuItems for 'About NNI'
            JMenuItem about = new JMenuItem("About NNI");
            about.addActionListener(new About());
            helpmenu.add(about);
    //Adding helpmenu to the menubar
            mb.add(helpmenu);//adding helpmenu 4

    //Add MenuBar, an intermediate container and layout manager to frame.
            setJMenuBar(mb);
    //Create different cards to be shows in same cardlayout
            card.setLayout(cardlayout);
            card.add(welcomePanel, welcomePanel.getKey());
            card.add(resultsPanel, resultsPanel.getResultKey());
            card.add(displayPanelsLoadedPanel,
                     displayPanelsLoadedPanel.getDisplayPanelsLoadedKey());
            getContentPane().add(card,BorderLayout.CENTER);
            cardlayout.show(card, welcomePanel.getKey());
            setDefaultCloseOperation(EXIT_ON_CLOSE);
}
//Defining the class for filtering initial file reading
```

```
class MyFileFilter implements FileFilter {
    //method of abstract class FileFilter that has to be defined
    public boolean accept(File file){
        if (file.isDirectory())
            return false; // ignore if this is a directory
        // get the name of the files
        String fileName = file.getName();
        // if filename ends with PFsuffix then accept it
        if (fileName.endsWith("param.txt")) return true;
        return false; // otherwise, ignore it
    }
}

//Defining class EmptyCanvas
class EmptyCanvas extends JPanel{
    public Dimension getPreferredSize(){
        return new Dimension(1000, 800);
    }
}
//Defining class for Confirmation of added new predictor
class DisplayPanelsLoaded extends JPanel{
    private String str = "Selected predictor(s) for the current run are: ";
    String message=
      "Select another predictor or load input from Analysis menu.";
    //DisplayPanelsLoaded constructor.
    DisplayPanelsLoaded(){
        JLabel label1, label2, space;
        JTextArea predtxt;
        setLayout(new GridLayout(2,2));
        label1 = new JLabel(str, JLabel.CENTER);
        space = new JLabel("", JLabel.CENTER);
        space.setBackground(Color.orange);
        space.setOpaque(true);
        label2 = new JLabel(message, JLabel.CENTER);
        label1.setFont(new Font("Times-Roman", Font.BOLD, 17));
        label1.setBackground(Color.orange);
        label1.setForeground(Color.red);
        label1.setOpaque(true);
        predtxt = new JTextArea();
        //Collect the titles of the currently load PF objects onto s
        String s = "";
        int len = pffiles.size();
        for (int i=len-1; i>=0; i--) {
            s= s + ((ParameterFile)pffiles.get(i)).getHead()+"\n";
        }
        predtxt.append(s);
        //Add labels to the JPanel.
```

```
            setBackground(Color.white);
            setBackground(Color.white);
            add(label1);
            add(space);
            add(predtxt);
            add(label2);
        }

        //Method for refreshing panel
        public void showIt(){
            displayPanelsLoadedPanel.removeAll();
            displayPanelsLoadedPanel.add(this);
            displayPanelsLoadedPanel.validate();
            cardlayout.show(card,
                    displayPanelsLoadedPanel.getDisplayPanelsLoadedKey());
        }

        //Method for calling the String id for the cards.
        public String getDisplayPanelsLoadedKey(){
            return str;
        }
    }
    //Defining class for deleting the latest loaded Prediction server
    class MinusPredictor implements ActionListener{
        JDialog dialog = new JDialog();

        public void actionPerformed(ActionEvent e){
            try
                {
                 //Check that ParameterFile object is loaded
                 System.out.println("This is within minuspredictor!");
                 if (pffiles.isEmpty()){
                    JOptionPane.showMessageDialog(dialog,
                            "No parameter file to remove. ", null,
                             JOptionPane.ERROR_MESSAGE  );
                    throw new NnIException("Lacking ParameterFile object");
                 } else {
                     ParameterFile pf =
                         (ParameterFile)pffiles.get(pffiles.size()-1);
                     String head = pf.getHead();
                     pffiles.remove(pffiles.size()-1);
                     DisplayPanelsLoaded doit3 = new DisplayPanelsLoaded();
                     doit3.showIt();
                     JOptionPane.showMessageDialog(dialog,
                            "The latest selected Predictor: \n" +head+
                           " has been deleted");
                 }
                } catch (NnIException d4){
                    System.err.println(d4.getMessage());
                } catch (Exception ex) {
                    System.err.println(
                         "Deleting a loaded Predictor raised exception");
                    JOptionPane.showMessageDialog(null,
                         "Problem deleting "+ "a loaded Predictor.", null,
                         JOptionPane.ERROR_MESSAGE );
                }
        }
    }

    //Defining class for restarting NNI
    class Resets implements ActionListener{
        JDialog dialog = new JDialog();

        public void actionPerformed(ActionEvent e){
            try
                {//Check that ParameterFile, input data and Result
                 // objects exist.
                 System.out.println("Welcome to Restarts");
                 td=null;
                 int len = pffiles.size();
                 for (int i=len-1; i>=0; i--) {
                     pffiles.remove(i);
                 }
                 for (int i=allResults.size()-1; i>=0; i--) {
                     allResults.remove(i);
                 }
                 cardlayout.show(card, welcomePanel.getKey());
                } catch (Exception d10) {
                    System.err.println(
                       "Restarting NNI raised exception "+ d10);
                    d10.printStackTrace();
                    JOptionPane.showMessageDialog( null, "Problem clearing "+
                                                   "the system.", null,
                                                   JOptionPane.ERROR_MESSAGE );
                }

        }
    }
    //Defining class used to quit the NNI GUI.
    class Quits implements ActionListener {
        public void actionPerformed (ActionEvent e){
            System.exit(0);
        }
    }
```

```
//Defining class for loading parameter files
class GetParam implements ActionListener {
    String str;
    ParameterFile p;
    JDialog dialog = new JDialog();

    GetParam(String str){
        this.str = str;
    }
    public void actionPerformed (ActionEvent e) {
        System.out.println("Someone is knocking on "
                            +((JMenuItem)e.getSource()).getText());
        try
            {        // Create a ParameterFile object
                p = new ParameterFile(user, str);
                if (td != null ){
                    JOptionPane.showMessageDialog(dialog,
                        "You cannot select predictors once "+
                        "input data has been loaded. Consider File " +
                        "-> Reset NNI.",
                        null, JOptionPane.ERROR_MESSAGE);
                    throw
                      new NnIException("ParameterFile object is null.");
                } else
                    {
                     //Next, check if the ParameterFile arrayList is empty
                     if(pffiles.isEmpty()){
                         pffiles.add(p);
                         System.out.println("Parameter file added");
                         DisplayPanelsLoaded doit2 =
                            new DisplayPanelsLoaded();
                         doit2.showIt();
                     } else {
                      // If not, extract info about the alphabet
                      // used by the first
                      // network object within the already
                      // loaded ParameterFile.
                      System.out.println("Testing if Parameter file is"+
                                           "added");
                      ParameterFile pff = (ParameterFile)(pffiles.get(0));
                      String inhead = pff.getHead();
                      Network inplace = (Network)(pff.getfirstNetwork());
                      String strin = inplace.getInputalph();
                      System.out.println(strin);
                      boolean[] loadedNet = Compute.getUsedcode(strin);
                      String newhead = p.getHead();
```

```
          Network incoming =(Network)(p.getfirstNetwork());
          String newstrin = incoming.getInputalph();
          System.out.println(newstrin);
          boolean[] newNet = Compute.getUsedcode(newstrin);
          boolean seenBefore = inhead.equals(newhead);
          for (int i=1; i<pffiles.size();i++){
            if ((((ParameterFile)(pffiles.get(i))).getHead()).equals(newhead)) {
                seenBefore = true;
            }
          }
          if(Arrays.equals(loadedNet, newNet) && !seenBefore ){
                pffiles.add(p);
                System.out.println("Another Parameter file "+
                                   "was loaded");
                DisplayPanelsLoaded doit2 =
                   new DisplayPanelsLoaded();
                doit2.showIt();
            } else {
                JOptionPane.showMessageDialog(dialog,
                  "Your newly selected "+
                  "Predictor is unacceptable. "+
                  "\nIt has been loaded before "+
                  "OR uses alphabet that is " +
                  "different from that "+
                  "previously loaded.", null,
                  JOptionPane.ERROR_MESSAGE  );
                throw new NnIException("New Predictor is "+
                                  "incompatible with earlier"+
                                  "choice.");
            }
         }
        }
} catch (NnIException d){
    System.err.println(d.getMessage());
} catch (IOException d2){
    System.err.println(
       "Could not open file for input: " + str);
    JOptionPane.showMessageDialog(null,
       "Could not open file '"+ str +"' for input.",
        null, JOptionPane.ERROR_MESSAGE );
} catch (Exception d1) {
    System.err.println(
       "Creating ParameterFile object raised exception");
    JOptionPane.showMessageDialog( null, "Problem loading "+
                                   "Predictor Server.", null,
                                   JOptionPane.ERROR_MESSAGE );
}
```

```
    }
}

//Defining class for basic help info
class Basics implements ActionListener {
    public void actionPerformed (ActionEvent e){
      try
      { String msg= "Neural Network Interpreter (NNI) reads  \n"+
         "as input one or more predictors and \n" +
         "a single data sequence in FASTA format. \n" +
         "The output is the result of analysing the data sequence\n"+
         "using the predictors.\n\n" +
         "Use the Predictor menu to select the predictors \n" +
         "that you want to use on the data. \n\n" +
         "The predictor names are the same with those on "+
         "the CBS web page. \n" +
         "So consult the CBS web page at "+
         "http://www.cbs.dtu.dk/services/\n"+
         "if in doubt of which predictor to use.\n\n" +
         "Use the Analysis -> Load Input menu and load "+
         "your your test data \n"+
         "from file. The selected predictors are immediately "+
         "applied to the data \n "+
         "and the result is shown as both a text sequence and a "+
         "graph.\n\n" +
         "Use File -> Reset NNI if you want to repeat the process with "+
         "new input data or additional predictors.\n\n"+
         "Note that all NNI files should be kept in the same folder:\n\n"+
         user.getAbsoluteFile().toString();
      JOptionPane.showMessageDialog( null, msg);
      } catch (Exception dd1) {
          JOptionPane.showMessageDialog(null, "Problem with Basics", null,
                                        JOptionPane.ERROR_MESSAGE);
      }
    }
}
//Defining class About NNI information
class About implements ActionListener {
    public void actionPerformed (ActionEvent e){
        try
            { String msg=
              "Neural Network Interpreter (NNI) was  \n"+
              "written by Joan Campbell-Tofte (joan@itu.dk) in partial \n"+
              "fulfilment of the requirements of the degree of \n"+
              "M.Sc. in Information Technology (Software Development).\n"+
              "Supervisor: Prof. Peter Sestoft, KVL and ITU\n"+
              "Consultant: Anders Gorm Pedersen, Center for Biological\n"+
              "Sequence analysis, Technical University of Denmark." ;
              JOptionPane.showMessageDialog( null, msg);
            } catch (Exception dd1) {
                JOptionPane.showMessageDialog(null,
                   "Problem with About", null,
                   JOptionPane.ERROR_MESSAGE);
            }
    }
}

//Defining class for presenting info on FASTA format
class Fasta implements ActionListener {
    public void actionPerformed (ActionEvent e){
        try
            { String msg=
              "Specify the input sequences intended for processing by \n"+
              "browsing your computer and selecting a file in " +
              "FASTA format.\n\n"+
              "Example of two sequences in FASTA format:\n"+
              ">seq1\n" +
              "ACGATGATCGATGCTGAGCTAGCGCTCGCT\n"+
              ">seq2 \n"+
              "ASQKRPSQRHGSKYLATASTMDHARHGFL\n\n"+
              "Sequences can be submitted in upper or lower case. "+
              "For more information about FASTA format, see \n\n"+
              "http://www.ncbi.nlm.nih.gov/ BLAST/fasta.html." ;
              JOptionPane.showMessageDialog( null, msg);
            } catch (Exception dd3) {
                JOptionPane.showMessageDialog(null,
                   "Problem with Fasta", null, JOptionPane.ERROR_MESSAGE);
            }
    }
}

//Defining class for choosing & loading input data.

class LoadData implements ActionListener {
    String pf_list= ("");
    JDialog dialog = new JDialog();

    public void actionPerformed (ActionEvent e){
        try
        {
          if (pffiles.isEmpty()){
             JOptionPane.showMessageDialog(dialog,
                "Sorry, you cannot load input data "+
                "\nwithout having loaded a Predictor." ,
                null,JOptionPane.ERROR_MESSAGE);
             throw new NnIException("Predictor not yet loaded");
          } else if (!allResults.isEmpty()){
              JOptionPane.showMessageDialog(dialog,
                 "Sorry, but input data is already loaded.\n"+
                 "To load new input data, \n"+
                 "first select Reset from the File menu.", null,
                 JOptionPane.ERROR_MESSAGE);
              throw new NnIException("Input data already loaded");
          } else {
              //Create a filechooser which opens to data directory
              final JFileChooser fc = new JFileChooser(nni);
              int returnVal = fc.showOpenDialog(NnIFrame.this);
              if (returnVal == JFileChooser.APPROVE_OPTION){
                  File infile = fc.getSelectedFile();
                  td = new TestData(infile, pffiles);
                  //Analyse loaded input data
                  System.out.println("The object TestData was built");
                  Iterator pfilIter = pffiles.iterator();
                  System.out.println("Entering the while loop to select"+
                                     "ParameterFile");
                  while (pfilIter.hasNext()){
                      ParameterFile pf = (ParameterFile)pfilIter.next();
                      System.out.println("First PF is selected.");
                      ArrayList totResults = Result.analyseData(td, pf);
                      double[][] res = Result.setNetOutput(totResults, pf);
                      Result finalres = new Result(res);
                      allResults.add(finalres);
                      System.out.println("A full Prediction Server was"+
                                         " run");
                  }
                  ShowResults doit = new ShowResults();
              }
          }
        } catch (NnIException d){
            System.err.println(d.getMessage());
        } catch (Exception dd1) {
            System.err.println("Input data input error within LoadData");
            JOptionPane.showMessageDialog(null,
               "Problem Reading or getting to "+
               "file within Load data.", null, JOptionPane.ERROR_MESSAGE);
        }
    }
}

//Defining the class ShowResults for controlling presentation of results
class ShowResults extends JPanel {
```

```
private String str = "Results of analysis";
private JTextArea saver = new JTextArea();

ShowResults(){
    /*Declaring panels to be used for displaying text and graphics */
    JPanel displayArea = new JPanel();
    displayArea.setLayout(
       new BoxLayout(displayArea, BoxLayout.Y_AXIS));
    JTextArea txtArea;

    /*Clear results panel of stuff from last display in this session so
      it is ready to accept new results*/
    if (pffiles.size()>0) {
        resultsPanel.removeAll();
        resultsPanel.setLayout(new BorderLayout());
    }

    //for-loop for running through the ArrayList
    for (int i = 0; i < pffiles.size(); i++){
        JPanel panel1 = new JPanel();
        panel1.setLayout(new BoxLayout(panel1, BoxLayout.Y_AXIS));
        /* Collecting info from input data object for
           result presentation*/
        String datahead = td.getHeading();
        String testdata = td.getInd();
        /*Next, get information from Results and ParameterFile objects
          Note that pffiles.size() = allResults.size();*/
        ParameterFile currpF = (ParameterFile)pffiles.get(i);
        String server = currpF.getHead();
        String outAlph = currpF.getfirstNetwork().getOutputalph();
        String title = "Results of analysing " +datahead+
            "\n on Predictor: "+server;
        Result currRes = (Result)allResults.get(i);
        double[][]res = currRes.getNetOutput();
        txtArea = new JTextArea(title);
        txtArea.setFont(new Font("Courier", Font.BOLD, 12));
        String resString = "";
        for (int j=0; j<res.length; j++){
            if ( res[j][0] < 0.5 )
                resString = resString + outAlph.charAt(1);
            else
                resString = resString + outAlph.charAt(0);
        }
        txtArea.append("\n Sequence  :"+testdata);
        txtArea.append("\n Prediction:"+resString+"\n");
        saver.append(txtArea.getText()+"\n\n");
        panel1.add(txtArea);
```

```
            Display disp = new Display(res);
            panel1.add(disp);
            displayArea.add(panel1);

        }
        /*Adding displayArea now containing all the test graphics results
          to display area*/
        if (pffiles.size()>0) {
            resultsPanel.add(displayArea);
            resultsPanel.validate();
            cardlayout.show(card,  resultsPanel.getResultKey());
        }
    }
    //Method for extracting the result in text format for the class Saves
    public String getTextResults() {
        return saver.getText();
    }
    //Method for calling the String id for the cards.
    public String getResultKey(){
        return str;
    }
}
//Defining class for handling the saving of the results of analysis to file
class SaveData implements ActionListener {
    JDialog dialog = new JDialog();
    public void actionPerformed (ActionEvent e) {
        try
            {if (pffiles.isEmpty()){
                JOptionPane.showMessageDialog(dialog,
                   "Sorry, No results to be saved yet",
                   null, JOptionPane.ERROR_MESSAGE);
                throw new NnIException("No results");
            } else if (allResults.isEmpty()){
                JOptionPane.showMessageDialog(dialog,
                      "Sorry, No results yet.", null,
                      JOptionPane.ERROR_MESSAGE);
                throw new NnIException("No results");
            } else {
                JFileChooser chooser = new JFileChooser(nni);
                chooser.setDialogType(JFileChooser.SAVE_DIALOG);
                int result = chooser.showDialog(null, "Save");
                if (result == JFileChooser.APPROVE_OPTION) {
                    FileOutputStream fos =
                        new FileOutputStream(chooser.getSelectedFile());
                    ShowResults doit2 = new ShowResults();
                    String text = doit2.getTextResults();
                    byte data[] = text.getBytes();
                    fos.write(data);
                    fos.close ();
                }
            }
            } catch (NnIException d){
                System.err.println(d.getMessage());
            } catch (Exception ex) {
                System.err.println("Error while writing");
            }

        }
    }


    //Defining the contentPane of Welcome page
    protected static class Welcome extends JPanel{
        private String str = "Welcome to the Neural Network Interpreter";
        JLabel label1, label2, label3, space;
        Welcome () {
            ImageIcon icon = new ImageIcon("inter_big_logo.gif");

            setLayout(new GridLayout(6,1)); //6 rows, 1 column, for aesthetics

            label1 = new JLabel(str, JLabel.CENTER);
            label1.setFont(new Font("Times-Roman", Font.BOLD, 17));
            label1.setBackground(Color.orange);
            label1.setForeground(Color.red);
            label1.setOpaque(true);
            space =  new JLabel("", JLabel.CENTER);
            label2 = new JLabel("You can now select the Prediction servers"+
                                " you wish to use " +
                                "from the Predictor menu",
                                JLabel.CENTER);
            //Set the position of the text, relative to the icon:
            label2.setVerticalTextPosition(JLabel.TOP);
            label2.setHorizontalTextPosition(JLabel.CENTER);
            label2.setIconTextGap(80);

            // logo
            label3 = new JLabel("",
                                icon,
                                JLabel.CENTER);

            //Add labels to the JPanel.
            setBackground(Color.white);
            add(label1);
            add(space);
            add(label2);
            add(space);
            add(space);
            add(label3);
        }
        //Method for calling the String id for the cards.
        public String getKey(){
            return str;
        }
    }
    //Defining the contentPane for the Graphics object.
    class Display extends JPanel{
        //Background color and data objects to be used
        final Color bg = Color.white;
        double[][] res;

        Display(double[][] res){
            setLayout(new BorderLayout());
            setBackground(bg);
            this.res = res;
            int preferredWidth = 10*res.length;
            if (preferredWidth>1000)
                preferredWidth=1000;
            setPreferredSize(new Dimension(preferredWidth, 300));
        }
        //Overriding the paintComponent()
            public void paintComponent(Graphics g) {
                super.paintComponent(g);
                //Cast Graphics to the more versatile Graphics2D
                Graphics2D g2d = ( Graphics2D) g;
                //Declare Device space
                int margin = 50;
                int gwidth = getWidth()-2*margin;
                int gheight = getHeight()-2*margin;
                g2d.setColor( Color.black );
                // Calculate user space, from testdata.length()(x)
                // & netresult[0][](y)
                for (int q = 0; q < res.length; q++){
                    g2d.drawLine( margin+q*gwidth/res.length, gheight,
                                  margin+q*gwidth/res.length, (int)
                                  (gheight - res[q][0]*2.0/3.0*gheight));
                    g2d.setColor( Color.red );
                    g2d.drawLine(margin+q*gwidth/res.length,
                                 gheight-(gheight/3),
                                 margin+q*gwidth/res.length,
                                 gheight-(gheight/3));
                }
                g2d.setColor( Color.blue );
                g2d.drawRect(margin, margin, gwidth, gheight-margin);
                g2d.drawString("Sequence position",
                               margin+gwidth/5, gheight+30);
                for (int i = 0; i < res.length; i=i+5){
                    g2d.drawString(""+i+"", margin+i*gwidth/res.length,
                                   gheight+20);
                    g2d.drawLine(margin+i*gwidth/res.length, gheight,
                                 margin+i*gwidth/res.length, gheight+10);
                }
                for (int k =0; k<15; k=k+5)
                    g2d.drawString(""+(double)k/10+"", 30,
                                   gheight-(gheight*k/15));
                AffineTransform at1 = new AffineTransform();
                at1.setToRotation(-Math.PI/2.0, 20, gheight-(gheight*1/3));
                g2d.transform(at1);
                g2d.drawString("Prediction value", 20, gheight-(gheight*1/3));

            }

    }
}
```

Appendix C

File NeuNetInterpreter.java

```
/*@(#)NeuNetInterpreter.java v2.2 02/07/2002
 NeuNetInterpreter.java starts the system by calling the NNI GUI.*/

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;

public class NeuNetInterpreter extends NnIFrame{


    public static void main(String args[]) {

        System.out.println("The Neural Network Interpreter starts here");

        try {
            UIManager.setLookAndFeel(
                                 UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { }
        NnIFrame jf = new NnIFrame();
        jf.pack();
        jf.show();

    }
}
```

Appendix D

File ParameterFile.java

```
/* @(#)ParameterFile.java v2.3 06/10/2002
*Code for generating the parameter file (PF)object which characterizes
*the CBS neural network-based predictors. This program
*also contains all the accessor methods for the PF fields*/

import java.io.*;
import java.util.*;
import javax.swing.*;

public class ParameterFile {

    private ArrayList netwk;
    private String nameString, head, sub;
    private boolean acceptsX, windowPartial, averagesResult;

    ParameterFile (){
    }

    ParameterFile (ArrayList netwk, String nameString,
                   String head, boolean acceptsX,
                   boolean windowPartial, String sub, boolean averagesResult){
        this.nameString = nameString;
        this.netwk = netwk;
        this.head = head;
        this.acceptsX = acceptsX;
        this.windowPartial = windowPartial;
        this.sub = sub;
        this.averagesResult = averagesResult;
    }

    //Constructor 3 which takes a file as argument
    ParameterFile (File dir, String st) throws NnIException, IOException {

    String nameString = "";
        String head = "";
        boolean acceptsX = true;
        boolean windowPartial = true;
        boolean averagesResult = true;
        String sub = "";
        ArrayList netwk = new ArrayList();
        ParameterFile pf = null;

        //Begin reading from input file;
    File f = new File(dir, st);
        FileReader inp = new FileReader (f);
        BufferedReader infile = new BufferedReader(inp);

        while (infile.ready()) { //while loop for going through whole text
            String tstrin = infile.readLine();
            if (tstrin.startsWith("Predictor")){
                StringTokenizer tst = new StringTokenizer(tstrin, ":");
                tst.nextToken(); //skipping first token on the line.
                nameString = (tst.nextToken()).trim();
                this.nameString = nameString;
                System.out.println(nameString);
            }
            if (tstrin.startsWith("Output")){
                head = nameString + " which runs the "+tstrin.substring
                                             (tstrin.indexOf(":"));
                this.head = head;
                System.out.println(head);
            }
            if (tstrin.startsWith("Allows")) {
                StringTokenizer tst1 = new StringTokenizer(tstrin, ":");
                tst1.nextToken(); //skipping first token on the line.
                String st1 = (tst1.nextToken()).trim();
                System.out.println(st1);
                        if (st1.equals("YES")){
                            acceptsX = true;
                        } else {
                            acceptsX = false;
                        }
                this.acceptsX = acceptsX;
                System.out.println("So predictor accepts x is "+acceptsX);
            }
            if (tstrin.startsWith("Permits")) {
                StringTokenizer tst2 = new StringTokenizer(tstrin, ":");
                tst2.nextToken(); //skipping first token on the line.
                String st2 = (tst2.nextToken()).trim();
                System.out.println(st2);
                        if (st2.equals("YES")){
                            windowPartial = true;
                        } else {
                            windowPartial = false;
                        }
                    this.windowPartial = windowPartial;
                    System.out.println("Here, windowpartial is "+windowPartial);
                }
                if (tstrin.startsWith("Synapse")) {
                    StringTokenizer tst3 = new StringTokenizer(tstrin, ":");
                    System.out.println(tst3.nextToken());
            //Line above skips first token, now extract synapse files.
                    StringTokenizer tst4 =
                                  new StringTokenizer(tst3.nextToken(), " ");
                    while (tst4.hasMoreTokens()){
                        String st4 = (tst4.nextToken()).trim();
                        System.out.println(st4);
           //Using the synapse file, create a neural network
                        try{
                                Network nwk = new Network(dir, st4);
                                System.out.println("Was network created?");
           //store neural network within ParameterFile’s network arraylist.
                                netwk.add(nwk);
                        } catch (IOException e){
                            e.printStackTrace(System.err);
                            System.err.println("Could not open file for input: "
                                               + st4);
                            JOptionPane.showMessageDialog(null,
                                    "Could not open file ’"+ st4 +"’ for input.",
                                    null,
                                    JOptionPane.ERROR_MESSAGE );
                            throw new NnIException("Failed to load synapse file.");
                         }
                    }
                    this.netwk = netwk;
                }

                if (tstrin.startsWith("Sequence pattern")){
                    StringTokenizer tst5 = new StringTokenizer(tstrin, ":");
                    tst5.nextToken();
                            if (tst5.hasMoreTokens()==true){
                                String st5= (tst5.nextToken()).trim();
                                sub = st5;
                            } else {
                                sub = "";
                            }
                    this.sub = sub;
                    System.out.println("Talking about subst " +sub);
                }

                if (tstrin.startsWith("Averages")) {
                    StringTokenizer tst6 = new StringTokenizer(tstrin, ":");
                    tst6.nextToken(); //skipping first token on the line.
                    String st6 = (tst6.nextToken()).trim();
                    System.out.println(st6);
                            if (st6.equals("YES")){
                                averagesResult = true;
                            } else {
                                averagesResult = false;
                            }
                    this.averagesResult = averagesResult;
                    System.out.println("AveragesResults should be "
                                       + averagesResult);
                }
            }
            infile.close();
        }
            //Method for extracting nameString for the PF object
            public String getNameString(){
                    return nameString;
        }
        //Method for extracting title for ParameterFile object
        public String getHead(){
                    return head;
        }

        //Method for extracting window filling information.
        public boolean getWindowPartial (){
                    return windowPartial;
        }

        //Method used for extracting target substring used in searching seq.
        public String getSub (){
                    return sub;
        }

        //Method for extracting ArrayList containing the network objects
        public ArrayList getNetObject(){
                    return netwk;
        }

        //Method for extracting the first Network from PF
        public Network getfirstNetwork(){
                    ArrayList networks = getNetObject();
                    Network firstnet = (Network)(networks.get(0));
```

```
                    return firstnet;
    }

    //Method for extracting the boolean x from PF
    public boolean getX(){
                return acceptsX;
    }

    //method for obtaining the alphabet content of the network
    public String getAlphabet(){
        Network firstnet = getfirstNetwork();
        String fafa = firstnet.getInputalph();
                if (getX() == true)
                    fafa = fafa + ’x’;
                return fafa;
    }

    //Method for checking how the final predictor output is decided
    public boolean getAveragesResult(){
            return averagesResult;
    }

    //Method for checking Predictor type,i.e., nucleic or protein
    public boolean getPredictorType(){
            Network repNetwork = getfirstNetwork();
            String alphabet = repNetwork.getInputalph();
            if (Compute.isDNA(alphabet))
               return true; // if network’s inputalph isDNA
            return false; // otherwise, it is protein type!
   }
}
```

Appendix E

File Network.java

```
/* @(#)Network.java v2.3 06/10/2002
*Code for generating the neural network from a synapse file.
*Contains 3 constructors and method for accessing the Network’s fields.*/

import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.beans.*;
import java.awt.event.*;

public class Network {
    private String inalph;
    private String outputalph;
    private int windowSize;
    private int nI, nH, nO;
    private Neuron[] hidden;
    private Neuron[] output;

    Network (){
    }

        //Constructor (2) that takes the Network fields as argument.
    Network (String inalph, String outputalph, int windowSize, int nI, int nH,
             int nO, Neuron[] hidden, Neuron[] output){
        this.inalph = inalph;
        this.outputalph = outputalph;
        this.windowSize = windowSize;
        this.nI = nI;
        this.nH = nH;
        this.nO = nO;
        this.hidden = hidden;
        this.output = output;
    }
```

```
/*Network constructor (3) takes object file as argument and builds the
    network*/
    Network (File dir, String file) throws IOException {
    String inalph = "";
    String outputalph = "";
    int windowSize=0, nI=0, nH=0, nO=0;
    double wgts = 0.0, bias = 0.0;
    int NeuronCount = 0;
    Neuron ny=null, ny1=null;
    Neuron [] h;
    Neuron [] o;

    //Begin reading from input file;
    FileReader inpp = new FileReader (new File(dir, file));
    BufferedReader infile = new BufferedReader(inpp);
    StreamTokenizer tstream = new StreamTokenizer(infile);
    tstream.parseNumbers();
    tstream.wordChars(’:’, ’:’);
    tstream.eolIsSignificant(false);
    System.out.println( "So far, so good.");
    skipToken(tstream);
    skipToken(tstream);
    //read input alphabet for network
    tstream.nextToken();
    inalph = tstream.sval;
    this.inalph = inalph;
    System.out.println("alphabet for network is " + inalph);
    skipToken(tstream);
    //read output alphabet for network
    tstream.nextToken();
    outputalph = tstream.sval;
    this.outputalph = outputalph;
    System.out.println("output alphabet for network is " + outputalph);
    //read nI = tstream.nextToken();
    nI = getNumber(tstream);
    this.nI = nI;
    System.out.println("nI= " + nI);

    // read nH
    skipToken(tstream);
    skipToken(tstream);
    nH = getNumber(tstream);
    this.nH = nH;
    System.out.println("nH= " + nH);

    // read nO
skipToken(tstream);
skipToken(tstream);
nO = getNumber(tstream);
this.nO = nO;
System.out.println("n0= " + nO);
for (int i = 0; i<4; i++)
    skipToken(tstream);
//read window size
windowSize = getNumber(tstream);
this.windowSize= windowSize;
System.out.println("Window size is " + windowSize);
for (int i = 0; i<3; i++)
   skipToken(tstream);
// read and store all the weights and thesholds of the nH hidden neurons
this.hidden = new Neuron [nH];

for (int j=0; j<nH; j++){
    //Read the weights & bias values for j Neurons.
    double [] weights= new double [nI];
    for (int k=0; k<nI; k++){
        weights[k] = getDouble(tstream);
    }
    bias = getDouble(tstream);
    ny = new Neuron(weights, bias);
    NeuronCount ++;
    hidden[j]= ny;
}
System.out.println(NeuronCount + " so far.");

// read and store all the weights and bias values of the nO output
//neurons
this.output = new Neuron [nO];

for (int m=0; m<nO; m++){
    //Read the weights & bias values for m Neurons.
    double [] weights= new double [nH];
    for (int n=0; n<nH; n++){
        weights[n] = getDouble(tstream);
    }
    bias = getDouble(tstream);
    ny1 = new Neuron(weights, bias);
    NeuronCount ++;
    output[m]= ny1;
}
infile.close();
System.out.println(NeuronCount + " at the end.");
}
```

```
    /*Method for calculating output from network on receiving  input
      from one windowfull of input or corresponding to evaluation of
      one input string position*/
    public double[] computeOutput (double[] input){
     //input table is calculated from network’s total inalph*window size.
            double[] resH = new double[nH];
            double[] resO = new double[nO];

     //first compute the total output from hidden neuron layer into []resH
            for (int v=0; v<resH.length; v++){
                    resH[v] = hidden[v].calOutput(input);
            }
     //Next,compute the 2 outputs from the output neuron layer into []resO
     //per point on test data. Return this in an array of doubles - result.
            for (int u=0; u<resO.length; u++){
                    resO[u]= output[u].calOutput(resH);
            }

            return resO;
    }
    //skipToken() required in constructor nr. 3.
    public void skipToken(StreamTokenizer str){
    try {str.nextToken();
    //         System.out.println("skipping: " + str.sval);
    }
    catch (IOException id1)
        {System.out.println("Failed skipping token: end of stream");}
}
//getNumber() required in constructor nr. 3
public int getNumber(StreamTokenizer str){
    try {str.nextToken();
    //  System.out.println("getting integer from stream: " + str.nval);
    Double d = new Double(str.nval);
    return(d.intValue());
    }
    catch (IOException id2)
        {System.out.println("Failed getting number " + str.sval);
        return 0;}
}
//getDouble() required in constructor nr. 3.
public double getDouble(StreamTokenizer str){
    try {str.nextToken();
    // System.out.println("getting double from stream: " + str.nval);
    return(str.nval);
    }
    catch (IOException id3)
```

```
            {System.out.println("Failed getting double from stream: " +
                                str.sval);
            return 0;}
    }
        //Method for extracting windowSize of network
    public int getWindow(){
                return windowSize;
    }

    //method for extracting N (total number of alphabet) used in network.
    public int getN(){
                return nI/windowSize;

    }

    //Method for extracting output alphabet from network
    public String getOutputalph(){
                   return outputalph;
    }

    //Method for extracting input alphabet from network
    public String getInputalph(){
            return inalph;
    }
}
class Neuron {

    double[] weights;
    double bias;

    Neuron (){
    }

    Neuron (double[]weights, double bias){

        this.weights = weights;
        this.bias = bias;
    }
    public double calOutput (double[] inputs){
        double j = 0.0;
        double x = 0.0;
        double output = 0.0;
        for (int k=0; k<inputs.length; k++){
            j = j+inputs[k]*weights[k];
        }
        x = j+bias;
        output = 1/(1+Math.exp(-x));
        return output;
    }
}
```

Appendix F

File Compute.java

```
/* @(#)Compute.java v2.5 06/10/2002
 *Code for the Compute.java program that contains all the helper methods
 *assisting in the preparation, submission and and analysis of test data by
 *the network.*/

import java.io.*;
import java.util.*;

public class Compute  {//1

    Compute() {
    }

    //method for comparing alphabet used in Strings.
    //Generate boolean tables of used alphabet and compare the tables
    public static boolean[] getUsedcode (String s){
        boolean[] seen = new boolean[256];
        for (int j=0; j < s.length(); j++){
            seen[s.charAt(j)]=true;
        }
        return seen;
    }

    //method for investigating if a string consists of DNA or protein
    //See if 80% consists of ACG&T.
    public static boolean isDNA (String str){
        int count = 0;
        boolean dna = false;

        for (int n=0; n<str.length(); n++){
            switch (str.charAt(n)){
            case ’A’: case ’C’: case ’G’: case ’T’:
            case ’a’: case ’c’: case ’g’: case ’t’:
                count++;
        default:
        }
    }
    if (100 * count >= 80 * str.length()){
        dna = true;
    }
    return dna;
}

//Method for extracting index of individual alphabets in "inalph".
public static int getIndex (char ch, String str){
            int ind = 0;
            for (int i = 0; i < str.length(); i++){
                if (str.charAt(i)==ch)
                            ind = i;
              }
            return ind;
}
// Method for reversing the contents of a string
public static String revContent (String str){
    String newst = new StringBuffer(str).reverse().toString();
            return newst;
}
// Method for extracting a subarray of doubles, starting with index
// start, continuing to (but not including) index last from byte[].
public static double[] extract(byte[] targetarray, int start, int last) {
    double[] subarray = new double[last - start];
    for(int i = 0; i < subarray.length;  i++)
        subarray[i] = (double) targetarray[i + start];
    return subarray;
}

//Method for sparse encoding test data into N binary digits.
public static byte[] encodeTestData (String testd, ParameterFile pfile){
    //First extract necessary info from first network in pfile.
    Network firstnet = pfile.getfirstNetwork();
    //System.out.println("Within encodeTData() & network extracted");
    int numalph = firstnet.getN();
    String st = pfile.getAlphabet();
    int win = firstnet.getWindow();
    String input = revContent(testd);
    /*System.out.println("The input string after reversion is : "+
      "\n"+input);
      System.out.println("Have extracted stuff from pfile in
      encodeTData()");*/
    byte[] coded;
    int startfill;
```

```
        //Calculate the size for the array to hold all the encoded test data
        if (!pfile.getWindowPartial()){
            System.out.println(pfile.getWindowPartial());
            System.out.println(!pfile.getWindowPartial());
            System.out.println("Here, u cannot have null indices");
            int bytesize = input.length()*numalph;
            coded = new byte[bytesize];
            startfill = 0;
        } else {
            /*The array to hold all the encoded test data is larger by
              windowSize-1. This is to allow for examination of first &
              last test data elements
              System.out.println(pfile.getWindowPartial()+ " within else.");
              System.out.println(!pfile.getWindowPartial());*/
            System.out.println("Here, u can jolly well have null indices");
            int bytesize = (input.length() + (win-1))*numalph;
            coded = new byte [bytesize];
            startfill = ((win-1)/2)*numalph;
        }

        for (int i=0; i<input.length(); i++){
            char c1 = input.charAt(i);
            int num = getIndex(c1, st);
            for (int j=0; j<numalph; j++){
                coded[startfill] = (j==num ? (byte) 1:0);
                startfill++;
            }
        }

        return coded;
    }
}
```

Appendix G

File TestData.java

```
/*  @(#)TestData.java v2.2 06/10/2002.
*Code for generating the test data object which holds test
*sequences presented by user in the FASTA format.*/

import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.beans.*;
import java.awt.event.*;

public class TestData {
    private String heading;
    private String ind;
    JDialog dialog = new JDialog();

    TestData (){
    }

    TestData (String heading, String ind){
        this.heading = heading;
        this.ind = ind;
    }

    //3rd constructor which takes a user submitted file name as argument.
    TestData (File file, ArrayList pffiles)
                  throws NnIException, IOException{
        String heading = "";
        String ind = "";
      //Begin reading from input file;
        FileReader inp = new FileReader (file);
        BufferedReader infile = new BufferedReader(inp);
        String tstring = infile.readLine();
        if (!tstring.startsWith(">")){
```

```
        JOptionPane.showMessageDialog(dialog,
                                      "Wrong format for input data. "+
                            "\nPlease resubmit data in the FASTA format.",
                            null, JOptionPane.ERROR_MESSAGE);

        throw new NnIException("Wrong test data format");

    } else {
        heading = heading + tstring.substring(1);
        System.out.println("This is within testData constructor 3");
        System.out.println("This heading is: " + heading);
        while (infile.ready()){
            tstring = infile.readLine();
            String tst = tstring.trim();
            ind = ind + tst;
        }
        System.out.println("The test string itself is: " + ind);
        //Check if the test data is of same type as in alphabet
        //for the networks.
        ParameterFile parafile = new ParameterFile();
        Network net = new Network();
        parafile = (ParameterFile)(pffiles.get(0));
        net = (Network)(parafile.getfirstNetwork());
        System.out.println("Network extracted");
        boolean balph = Compute.isDNA(net.getInputalph());
        boolean talph = Compute.isDNA(ind);
        if (heading.equals("") || balph != talph){
            JOptionPane.showMessageDialog(dialog,
                           "Alphabet of input sequence conflicts with "+
                           "the alphabet of the Predictor.",null,
                           JOptionPane.ERROR_MESSAGE);

            throw new NnIException("No match between Predictor and"+
                                   "test data alphabet ");
        } else {
            this.heading = heading;
            this.ind = ind;
            infile.close();
        }
    }

}

//Method for extracting title for Testdata object
public String getHeading(){
    System.out.println("Entered getHeading");
    return heading;
    }
    //Method for extracting Testdata’s content
    public String getInd(){
        return ind;
    }
}
```

Appendix H

File NnIException.java

```
//  @(#)NniException.java v1.0 12/10/2002.

public class NnIException extends Exception {
        public NnIException (String msg){
                super(msg);
        }
}
```

Appendix I

File Result.java

```
/* @(#)Result.java v1.0 07/10/2002
 *Code for the Result.java program that generates and stores the result
 *from the analysis of test data by the network.*/

import java.io.*;
import java.util.*;

public class Result  {
    private double[][]netOutput;

    Result() {
    }

    Result (double[][] netOutput){
        this.netOutput = netOutput;
    }

    //Method for submitting window-size of test data as network input and
    //generating Result object.
    public static ArrayList analyseData (TestData td, ParameterFile pfile){
        System.out.println("Entered the method analyseData()");
        //Extracting test data from String field of TestData.
        String testSt = (td.getInd()).toUpperCase();
        String newString;
        //Then extract info required for driving analysis from ParameterFile
        //object
        Network firstnet = pfile.getfirstNetwork();
        int win = firstnet.getWindow();
        int n = firstnet.getN();
        int inputSize = win*n;
        //extract search substring
        String subst = (pfile.getSub()).toUpperCase();
        ArrayList netwks = pfile.getNetObject();
```

```
//Declaration of arrays to be used during analysis
byte[]byt = Compute.encodeTestData(testSt, pfile);
double[] subbyt = new double[inputSize];
double[] myResult;//output calculated from one input window
//2-dimensional array for holding results from individual networks
double[][] netResult = new double[testSt.length()][2];
//Make an ArrayList object to hold netresult obj from networks
ArrayList totResult = new ArrayList();

//Call an iterator to call the networks one at time on input
Iterator net = netwks.iterator();
while (net.hasNext()){
    Network currentNet = (Network)net.next();
    //       System.out.println("Have extracted Networks, etc");
    if (pfile.getWindowPartial()==true){
        String s = "";//Making the string to pad test data with.
        for (int pad = 0; pad < (win-1)/2; pad++)
            s = s + '-';
        newString = s + testSt + s;
        System.out.println("This is the padded string: " +newString);
    } else {
        newString = testSt;
        System.out.println("The string: "+newString+" is not padded");
    }
    for (int start=0; start<=newString.length()-win; start++){
        //current search window
        String searchWin = newString.substring(start, start+win);
        //sets point for sampling data in byt[]
        int b = (byt.length - inputSize) - (n * start);
        //Cond. statement to control extraction of input data for
        //network
        if(subst.equals("") || searchWin.startsWith(subst, (win-1)/2)){
            //Will happen for all windows for a & with  a few for b
            System.out.println(searchWin);
            //extraction of byt[] for network
            subbyt = Compute.extract(byt, b, b + inputSize);
            myResult = currentNet.computeOutput(subbyt);
            //Filling up netresult[][]
            if (pfile.getWindowPartial()==true){
                netResult[start]= myResult;
            } else {
                netResult[(start+(win-1)/2)] = myResult;
            }

        }

    }
```

```
        totResult.add(netResult);
        System.out.println("We got to calculate netResult alright");
    }
    return totResult;
}

//Method for calculating averaged or voted output from the networks
public static double[][] setNetOutput(ArrayList totResult,
                                      ParameterFile pfile){
        System.out.println("System running within setNetOutput()");
        // Extracting first netresult[][]
        // for use in defining Result object size.
        double[][] firstNetResult = (double[][])(totResult.get(0));
        double[][] r = new double[firstNetResult.length]
                                 [firstNetResult[0].length];
        int listSize = totResult.size();
        System.out.println("The Result object is going to be "
                           +firstNetResult.length+" long.");
        System.out.println(
          "Loop for running through the contents of all the tables");
        System.out.println("so"+pfile.getAveragesResult());
        for(int i = 0; i < r.length; i++){
        for(int j = 0; j < r[i].length; j++){
            if (pfile.getAveragesResult() == true){
                Iterator resIter = totResult.iterator();
                while (resIter.hasNext()){
                    r[i][j] = r[i][j] + ((double[][])resIter.next())[i][j];
                }
                r[i][j] = r[i][j]/listSize;
                System.out.println("This was an averaging predictor");
            } else {
                    System.out.println(
                          "This is inside the jury-predictor domain");
                    Iterator resIter = totResult.iterator();
                    int counterno = 0, counteryes = 0;
                    double tempno = 0.0, tempyes = 0.0;
                    while (resIter.hasNext()){
                        double currentVal =
                           ((double[][])resIter.next())[i][j];
                        if(currentVal < 0.5){
                            counterno++;
                            tempno = currentVal;
                        } else {
                            counteryes++;
                            tempyes = currentVal;
                        }
                    }
```

```
                        if(counterno > counteryes){
                            r[i][j] = tempno;
                        } else {
                            r[i][j] = tempyes;
                        }
                    }
                }
            }
            for (int p= 0; p<r.length; p++){
                for (int q=0; q<r[p].length; q++){
                    System.out.print("result[" + p + "]["+ q + "]= "
                                     + r[p][q] + "\n");
                }
            }
            return r;
        }

        //Method for extracting result from the Result object
        public double[][] getNetOutput(){
            return netOutput;
        }
    }
```

Appendix J

Source codes used for carrying out the unit tests

J.1 Testing the NnIFrame class: NnIFrame.java

```
/* @(#)NnIFrame.java v2 28/09/2002, Early version of NNI GUI used for
demonstration of unit tests carried out for the NnIFrame class.*/
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;

//Code for the basic NnIFrame GUI
public class NnIFrame extends JFrame {

        NnIFrame(){

                addWindowListener(new WindowAdapter() {
                                public void windowClosing (WindowEvent e){
                                        System.exit(0);}
                });

                setTitle("Neural Network Interpreter");

                JMenuBar mb = new JMenuBar();
                setJMenuBar(mb);

                JMenu filemenu = new JMenu("File");

                //Instantiating the ActionListener for uploading files
                Opens pen = new Opens();
                Load op = new Load();
        //The ’open’ menuItem for File menu.
        JMenuItem open = new JMenuItem ("Open file");
        open.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_O, ActionEvent.ALT_MASK));
        filemenu.add(open);
        open.addActionListener(pen);

        //The ’save’ menuItem for File menu.
        JMenuItem save = new JMenuItem ("Save as..");
        save.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_S, ActionEvent.ALT_MASK));
        filemenu.add(save);

        // Following code allows user to print in NNI
        JMenuItem print = new JMenuItem ("Print");
        print.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_P, ActionEvent.ALT_MASK));
        filemenu.add(print);

        // Following code allows user to quit NNI
        JMenuItem quit = new JMenuItem ("Quit NNI");
        quit.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_Q, ActionEvent.ALT_MASK));
        filemenu.add(quit);
        quit.addActionListener (new ActionListener() {
                public void actionPerformed (ActionEvent e){
                        System.exit(0);}
        });

        //Now add filemenu to menubar
        mb.add(filemenu);

        //Create networkmenu
        JMenu networkmenu = new JMenu("Network");

        JMenuItem nwk1 = new JMenuItem ("Load network");
        nwk1.setSelected(false);
        networkmenu.add(nwk1);

        //Adding the Load actionListener for ’nwk1’
        nwk1.addActionListener(op);


        JMenuItem nwk2 = new JMenuItem ("Delete network");
        nwk2.setSelected(false);
        networkmenu.add(nwk2);
        //Adding the Load actionListener for ’nwk2’
```

```
                nwk2.addActionListener(op);

                //Adding networkmenu to the menubar
                mb.add(networkmenu);

                //Create loadmenu
                JMenu loadmenu = new JMenu("Load Data");

                JMenuItem nucl = new JMenuItem ("DNA/RNA seq.");
                loadmenu.add(nucl);
                //Adding an Load actionListener for ’load DNA/RNA seq.’.
                nucl.addActionListener(op);

                JMenuItem prot = new JMenuItem ("Protein seq.");
                loadmenu.add(prot);
                //Adding the Load actionListener for ’load Protein seq’
                prot.addActionListener(op);

                //Adding loadmenu to the menubar
                mb.add(loadmenu);

                //Create helpmenu
                JMenu helpmenu = new JMenu("Help");

                //Adding helpmenu to the menubar
                mb.add(helpmenu);

        }
            //Defining class used to find and select files
            public static class Opens implements ActionListener {
                    private static File file = null;
                    private static NnIFrameny nnf;
                    public void actionPerformed (ActionEvent e){
                            //Create a filechooser
                            final JFileChooser fc = new JFileChooser();
                            int returnVal = fc.showOpenDialog(nnf);

                    }
            }
            //Defining class used to find and load synapse files
            public  class Load implements ActionListener {
                    JDialog dialog = new JDialog();
                    private  File infile = null;
                    private  NnIFrame nf;
                    public void actionPerformed (ActionEvent e){
                            //Create a filechooser
                            final JFileChooser fc = new JFileChooser();
                int returnVal = fc.showOpenDialog(nf);
        }
}
//Defining the contentPane of Welcome page
private static class Welcome extends JPanel{
        JLabel label1, label2, label3;

        Welcome () {
ImageIcon icon = new ImageIcon("starfshIcon.gif");

setLayout(new GridLayout(5,1));     //5 rows, 1 column, for aesthetics

label1 = new JLabel("Welcome to the Neural Network Interpreter", JLabel.CENTER);
label1.setFont(new Font("Times-Roman", Font.BOLD, 17));
label1.setBackground(Color.green);
label1.setForeground(Color.red);
label1.setOpaque(true);
label2 = new JLabel("You can now select the network you wish to use " +
                        "from the Network menu",
                    icon,
                    JLabel.CENTER);
//Set the position of the text, relative to the icon:
label2.setVerticalTextPosition(JLabel.TOP);
label2.setHorizontalTextPosition(JLabel.CENTER);
label2.setIconTextGap(40);

//Add labels to the JPanel.
add(label1);
add(label2);
setBackground(Color.white);

}
}

public static void main (String [] args){
         try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
} catch (Exception e) { }
//First things first, the Welcome window for NNI!
        NnIFrame frame = new NnIFrame();

        frame.setContentPane(new Welcome());
        frame.pack();
        frame.setSize(765, 690);
        frame.setVisible(true);
}
```

```
}
```

J.2 Testing the Network class: Network.java

```
/* @(#)Network.java v3 28/09/2002, Basic Network.java for unit
test.*/

import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.beans.*;
import java.awt.event.*;

public class Network {
    private String inalph;
    private String outputalph;
    private int windowSize;
    private int nI, nH, nO;
        private Neuron[] hidden;
    private Neuron[] output;

    Network (){         //The empty constructor(1)
   }

   //Constructor (2) that takes the Network fields as argument.
    Network (String inalph, String outputalph, int windowSize, int nI, int nH,
        int nO, Neuron[] hidden, Neuron[] output){
                this.inalph = inalph;
                this.outputalph = outputalph;
                this.windowSize = windowSize;
                this.nI = nI;
                this.nH = nH;
                this.nO = nO;
                this.hidden = hidden;
                this.output = output;
    }
        /*Network constructor (3) takes object file as argument and builds the
        network*/
        Network (String file) throws IOException {
        String inalph = "";
        String outputalph = "";
        int windowSize=0, nI=0, nH=0, nO=0;
        double wgts = 0.0, bias = 0.0;
        int NeuronCount = 0;
        Neuron ny=null, ny1=null;
        Neuron [] h;
Neuron [] o;

//Begin reading from input file;
FileReader inpp = new FileReader (file);
BufferedReader infile = new BufferedReader(inpp);
StreamTokenizer tstream = new StreamTokenizer(infile);
tstream.parseNumbers();
tstream.wordChars(’:’, ’:’);
tstream.eolIsSignificant(false);
System.out.println( "So far, so good.");
skipToken(tstream);
skipToken(tstream);
//read input alphabet for network
tstream.nextToken();
inalph = tstream.sval;
this.inalph = inalph;
System.out.println("alphabet for network is " + inalph);
skipToken(tstream);
//read output alphabet for network
tstream.nextToken();
outputalph = tstream.sval;
this.outputalph = outputalph;
System.out.println("output alphabet for network is " + outputalph);
//read nI = tstream.nextToken();
nI = getNumber(tstream);
this.nI = nI;
System.out.println("nI= " + nI);

// read nH
skipToken(tstream);
skipToken(tstream);
nH = getNumber(tstream);
this.nH = nH;
System.out.println("nH= " + nH);

// read nO
skipToken(tstream);
skipToken(tstream);
nO = getNumber(tstream);
this.nO = nO;
System.out.println("n0= " + nO);
for (int i = 0; i<4; i++)
    skipToken(tstream);
//read window size
windowSize = getNumber(tstream);
this.windowSize= windowSize;
System.out.println("Window size is " + windowSize);
```

```
for (int i = 0; i<3; i++)
   skipToken(tstream);
// read and store all the weights and thesholds of the nH hidden neurons
this.hidden = new Neuron [nH];

for (int j=0; j<nH; j++){
    //Read the weights & biasf values for j Neurons.
    double [] weights= new double [nI];
    for (int k=0; k<nI; k++){
        weights[k] = getDouble(tstream);
    }
    bias = getDouble(tstream);
    ny = new Neuron(weights, bias);
    NeuronCount ++;
    hidden[j]= ny;
}
System.out.println(NeuronCount + " so far.");

// read and store all the weights and bias values of the nO output neurons
this.output = new Neuron [nO];

for (int m=0; m<nO; m++){
    //Read the weights & bias values for m Neurons.
    double [] weights= new double [nH];
    for (int n=0; n<nH; n++){
        weights[n] = getDouble(tstream);
    }
    bias = getDouble(tstream);
    ny1 = new Neuron(weights, bias);
    NeuronCount ++;
    output[m]= ny1;
}
infile.close();
System.out.println(NeuronCount + " at the end.");
}

/*Method for calculating output from network on receiving  input from one
windowfull of input or corresponding to evaluation of one input string position*/
public double[] computeOutput (double[] input){
        //input table is calculated from network’s total inalph*window size.
        double[] resH = new double[nH];
        double[] resO = new double[nO];

        //first compute the total output from hidden neuron layer into []resH
        for (int v=0; v<resH.length; v++){
                resH[v] = hidden[v].calOutput(input);
        }
```

```
            //Next,compute the 2 outputs from the output neuron layer into []resO
            //per point on test data. Return this in an array of doubles - result.
            for (int u=0; u<resO.length; u++){
                    resO[u]= output[u].calOutput(resH);
            }

            return resO;
    }
    //skipToken() required in constructor nr. 3.
    public void skipToken(StreamTokenizer str){
    try {str.nextToken();
    //       System.out.println("skipping: " + str.sval);
    }
    catch (IOException id1)
        {System.out.println("Failed skipping token: end of stream");}
}
//getNumber() required in constructor nr. 3
public int getNumber(StreamTokenizer str){
    try {str.nextToken();
    //  System.out.println("getting integer from stream: " + str.nval);
    Double d = new Double(str.nval);
    return(d.intValue());
    }
    catch (IOException id2)
        {System.out.println("Failed getting number " + str.sval);
        return 0;}
}
//getDouble() required in constructor nr. 3.
public double getDouble(StreamTokenizer str){
    try {str.nextToken();
    // System.out.println("getting double from stream: " + str.nval);
    return(str.nval);
    }
    catch (IOException id3)
        {System.out.println("Failed getting double from stream: " + str.sval);
        return 0;}
}
    //Method for extracting windowSize of network
public int getWindow(){
            return windowSize;
}

//method for extracting N (total number of alphabet) used in network.
public int getN(){
            return nI/windowSize;

}
```

```
    //Method for extracting output alphabet from network
    public String getOutputalph(){
                return outputalph;
    }
    public static void main (String [] args){
        try{
        Network test = new Network (args[0]);
        System.out.println(test.getN());
        System.out.println(test.getOutputalph());
    } catch (IOException e) {
        e.printStackTrace(System.err);
    }
        System.out.println("Hallo test network");

        }
class Neuron {

    double[] weights;
    double bias;

    Neuron (){
    }

    Neuron (double[]weights, double bias){

        this.weights = weights;
        this.bias = bias;
    }
    public double calOutput (double[] inputs){
        double j = 0.0;
        double x = 0.0;
        double output = 0.0;
        for (int k=0; k<inputs.length; k++){
            j = j+inputs[k]*weights[k];
        }
        x = j+bias;
        output = 1/(1+Math.exp(-x));
        return output;
    }
  }
}
```

J.3 Testing the Test Data class: TData.java

```
/*  @(#)MakeTData.java v2.1 27/06/2002.
Code for generating the test data object which holds test
```

```
*sequences presented by user in the fasta format.*/

import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.beans.*;
import java.awt.event.*;

//import javax.swing.event.*;

class TestData extends JDialog{
  private String heading;
  private String ind;
  JDialog dialog = new JDialog();

  TestData (){ //empty constructor
  }

  TestData (String heading, String ind){ //Constructor (2)
    this.heading = heading;
    this.ind = ind;
  }
  TestData (String file) throws NnIException, IOException{
    String heading = "";
    String ind = "";

    //Begin reading from input file;
    FileReader inp = new FileReader (file);
    BufferedReader infile = new BufferedReader(inp);
    String tstring = infile.readLine();
    if (tstring.startsWith(">")==false){
      JOptionPane.showMessageDialog(dialog,
                           "Wrong format for test data. "+
                           "Please resubmit data in the fasta format.");

      throw new NnIException("Wrong test data format");

    } else {
      heading = heading + tstring;
      System.out.println("This is within testData constructor 3");
      System.out.println("This heading is: " + heading);
      while (infile.ready()== true){
        tstring = infile.readLine();
        String tst = tstring.trim();
        ind = ind + tst;
      }
      System.out.println("The test string itself is: " + ind);
      this.heading = heading;
      this.ind = ind;
      infile.close();
    }
  }
  //Method for extracting title for Test data object
  public String getHeading(){
    System.out.println("Entered getHeading");
    return heading;

  }
  //Method for extracting Test data’s content
  public String getInd(){
    return ind;
  }
  public static void main (String [] args){
    try{
      TestData lala = new TestData (args[0]);
      System.out.println(lala.getHeading());
      System.out.println(lala.getInd());

    } catch (NnIException d){
      System.err.println(d.getMessage());
    } catch (IOException e) {
      e.printStackTrace(System.err);
    }
  }
  //Defining NniException class
  class NnIException extends Exception {
    public NnIException (String msg){
      super(msg);
    }
  }
}
```

J.4 Testseq1.txt

```
>testseq 1
agatctgagagagagagagagtagatgatagatagcgctag
```

J.5 seq2.txt

```
>seq2
ASQKRPSQRHGSKYLATASTMDHARHGFLPRHRDTGILDSIGRFFGGDRGAPK
```

```
NMYKDSHHPARTAHYGSLPQKSHGRTQDENPVVHFFKNIVTPRTPPPSQGKGR
KSAHKGFKGVDAQGTLSKIFKLGGRDSRSGSPMARRELVISLIVES
```

J.6 Testseq3.txt

```
Testseq3
ASQKRPSQRHGSKYLATASTMDHARHGFLPRHRDTGILDSIGRFFGGDRGAPK
NMYKDSHHPARTAHYGSLPQKSHGRTQDENPVVHFFKNIVTPRTPPPSQGKGR
KSAHKGFKGVDAQGTLSKIFKLGGRDSRSGSPMARRELVISLIVES
```

J.7 OrdFASTAtext.txt

```
>OrdFASTAtext
The quick brown fox jumped over the lazy hound.
```

J.8 TranSparam.txt

```
Predictor name: TranS
Output header: Prediction of membrane proteins in Signal Transduction
Allows use of unknown symbol: YES
Permits use of partial windows: YES
Synapse file(s): TranSyn.txt
Sequence pattern for window selection:
Averages final predictor output: No
```

J.9 TranSyn.txt

```
TESTRUNID          IN: ACGT  OUT: sg
      12  LAYER:   1
       2  LAYER:   2
       2  LAYER:   3
     100 :ILEARN        3 :NWSIZE      -90 :ICOVER
     -0.22020     -0.24010     -0.17422     -0.25131     -1.36468
     -1.55053      3.48135     -1.45199     -0.35687     -0.48446
     -0.26066     -0.18385     -0.68584     -0.11751      0.03703
     -0.00989     -0.05937     -1.17373     -1.15452      2.46427
     -1.08576     -0.01796     -0.12046     -0.15830     -0.25301
     -0.74973      3.26438      2.19347     -2.59703     -3.46976
     -1.92930      2.57139
```

Bibliography

[1] Brazma, A., Parkinson, H., Schlitt, T. and Shojatalab, M. (2001) A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays. Available at: ⟨http://www.ebi.ac.uk/microarray/biology_intro.html⟩.

[2] The International Human Genome Mapping Consortium (2001) A physical map of the human genome. Nature 409:934–941.

[3] Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:443–453.

[4] Smith, A.F.M. and Waterman, M.S. (1981) Identification of common molecular subsequences. Journal of Molecular Biology, 147:195–197.

[5] Altschul, S.F. (1996) Local alignment statistics. Meth. Enzymol. 274:460–480.

[6] Baldi, P. and Brunak, S. (1998) Bioinformatics The Machine Learning Approach. The MIT Press, Cambridge, Massachusetts. London, England.

[7] Brunak, S. (1989) Linjedeling med et neuralt netværk. SAML 14: 55–74.

[8] Anders Gorm Pedersen (1997). Gene structure and gene expression. Ph.D thesis, Danish Technical University, Lyngby.

[9] Hertz, J., Krogh, A. and Palmer, R.G. (1991) Introduction to the theory of neural computation. Lecture notes volume I: Santa Fe Institute studies in the sciences of complexity. Addison-Wesley Publishing Company, Redwood City, California.

[10] Frö hlich, J. (1997) Neural networks in Java. Available at: ⟨http://rfhs8012.fh-regensburg.de/ saj39122/jfroehl/diplom/e–index.html⟩.

[11] Nilsson, N.J. (1996) Introduction to Machine learning Available at: ⟨http://robotics.stanford.edu/people/nilsson/mlbook.html.⟩.

[12] Olmsted, D.D. (1998) History and principles of neural networks from 1960 to present. Available at: ⟨http://www.neurocomputing.org/History/Neural_Network_History_2/body_neural_network_history_2.html⟩.

[13] Pedersen, A.G. and Nielsen, H. (1997) Neural network prediction of translation initiation sites in eukaryotes: perspectives for EST and genome analysis. ISMB: 5, 226-233.

[14] Beale, R. and Jackson, T. (1990) Neural Computing: an introduction. Adam Hilger, IOP Publishing, Bristol, Philadelphia and New York.

[15] Teuscher, C. Evolving artificial neural networks. (Available at:⟨http://lslwww.epfl.ch/pages/teaching/documents/eann.pdf⟩).

[16] Medler, D.A. and McClelland, J.L. (1999) Exploring the Role of Context and Sparse Coding on the Formation of Internal Representations; in M. Hahn & S. C. Stoness (Eds.), Twenty First Annual Conference of the Cognitive Science Society. Mahwah, NJ: Lawrence Erlbaum p393.

[17] Bansal, K., Vadhavkar, S. and Gupta, (1998) Brief application description: Neural Networks based forecasting techniques for inventory control applications. Data Mining and Knowledge Discovery. 2: 97–102.

[18] $Java^{TM}$ 2 Platform Standard Edition, v 1.4.0 API Specification. Available at: ⟨http://java.sun.com/j2se/1.4/docs/api/ ⟩.

[19] Jacobson, I., G. Booch, and Rumbaugh, J. (1999) Unified Software Development Process, Addison-Wesley.

[20] Wirfs-Brock, R., Wilkerson, B. and Weiner, L. (1990) Designing Object-Oriented Software, Prentice-Hall, New Jersey.

[21] McCabe, T. (1976) A Software Complexity Measure. IEEE Trans. Software Engineering, 14(6)308–320.

[22] Pressman, R.S. (2000) Software Engineering, A Practitioner's approach. McGraw-Hill Publishing Company, London, England. Burr Ridge IL, New York, USA.

[23] Jørgensen, U.L, and Jensen, T.S. (2002) Characterisation and Identification of cell cycle regulated genes from sequence derived protein features. M.Sc. thesis, Danish Technical University, Lyngby.

[24] Matthews, B.W. (1975) Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim. Biophys. Acta. 405: 442–451.

[25] Preece, J., Rogers, Y, Sharp, H. (2002) Interaction Design: Beyond human-computer Interaction. Wiley College. UK.

[26] Weigers, K. (1999) Software Requirements, Microsoft Press ISBN 0–7356–0631–5

[27] Pearson, W.R. and Lipmann, D.J (1988) Improved tools for biological sequence comparison. PNASU 85: 2444-2448.

[28] Eckel, B. (2000) Thinking in Java. Prentice Hall, Upper Saddle River, New Jersey. USA.

[29] Lieberman, B. (2001) UML activity diagrams: versatile roadmaps for understanding system behavior. Available at: ⟨http://www.therationaledge.com/content/apr_01/t_uml_bl.html⟩.